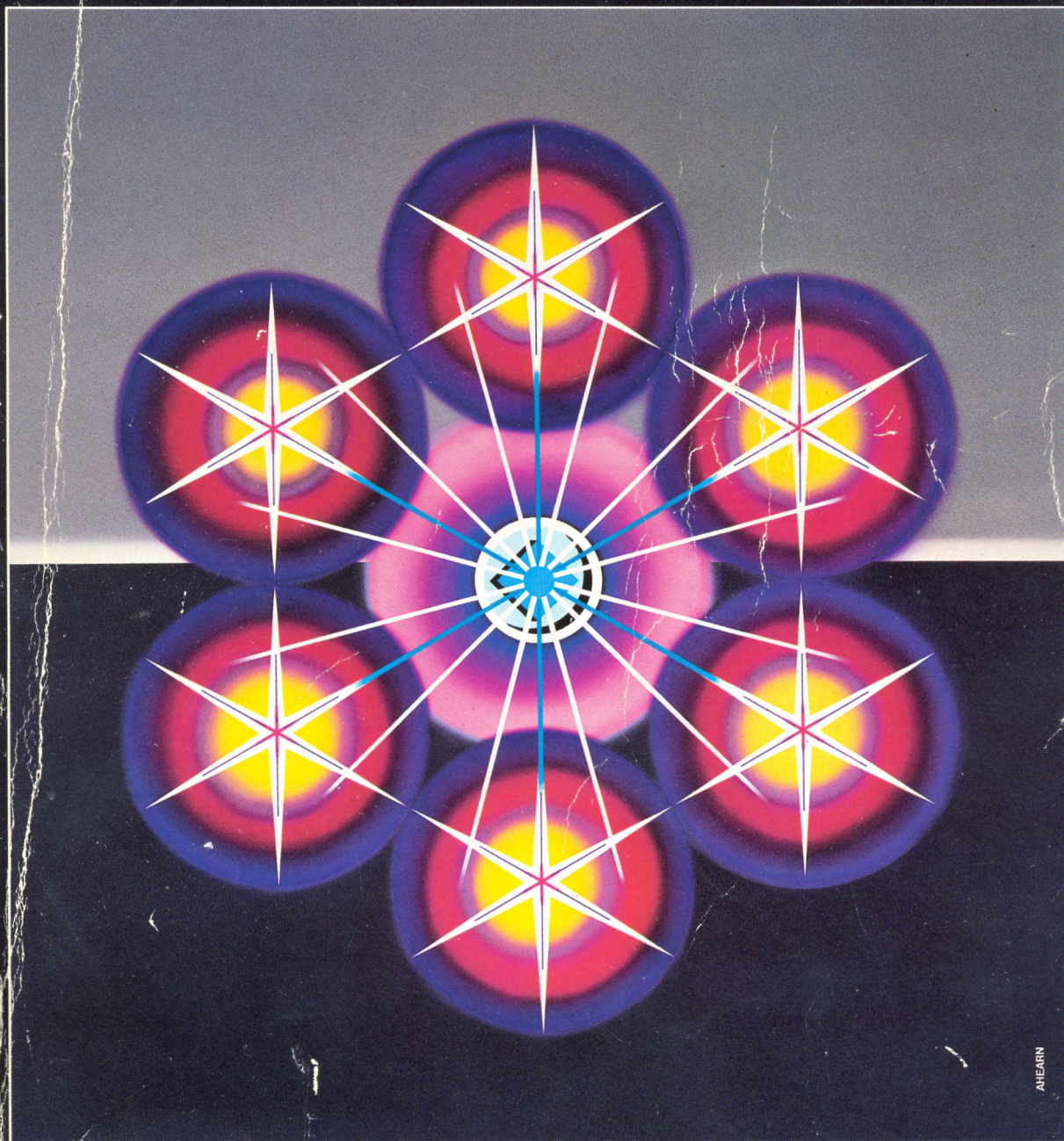


intel®

# Microcontroller Handbook



AREARN

Order Number: 210918-004



## LITERATURE

To order Intel Literature write or call:

Intel Literature Sales  
P.O. Box 58130  
Santa Clara, CA 95052-8130

Intel Literature Sales:  
(800) 548-4725  
Other Inquiries:  
(800) 538-1876

Use the order blank on the facing page or call our Toll Free Number listed above to order literature. Remember to add your local sales tax and a 10% handling charge for U.S. customers, 20% for Canadian customers.

### 1986 HANDBOOKS

Product Line handbooks contain data sheets, application notes, article reprints and other design information.

| NAME  | ORDER NUMBER | *PRICE IN<br>U.S. DOLLARS |
|---|--------------|---------------------------|
| <b>COMPLETE SET OF 9 HANDBOOKS</b><br>Get a 30% discount off the retail price of \$171.00               | 231003       | <b>\$120.00</b>           |
| <b>MEMORY COMPONENTS HANDBOOK</b>   | 210830       | <b>\$18.00</b>            |
| <b>MICROCOMMUNICATIONS HANDBOOK</b>   | 231658       | <b>\$18.00</b>            |
| <b>MICROCONTROLLER HANDBOOK</b>   | 210918       | <b>\$18.00</b>            |
| <b>MICROSYSTEM COMPONENTS HANDBOOK</b><br>Microprocessor and peripherals (2 Volume Set)                 | 230843       | <b>\$25.00</b>            |
| <b>DEVELOPMENT SYSTEMS HANDBOOK</b>   | 210940       | <b>\$18.00</b>            |
| <b>OEM SYSTEMS HANDBOOK</b>   | 210941       | <b>\$18.00</b>            |
| <b>SOFTWARE HANDBOOK</b>  | 230786       | <b>\$18.00</b>            |
| <b>MILITARY HANDBOOK</b>  | 210461       | <b>\$18.00</b>            |
| <b>QUALITY/RELIABILITY HANDBOOK</b>   | 210997       | <b>\$20.00</b>            |
| <b>PRODUCT GUIDE</b><br>Overview of Intel's complete product lines                                      | 210846       | <b>No charge</b>          |
| <b>LITERATURE GUIDE</b><br>Listing of Intel Literature  | 210620       | <b>No charge</b>          |
| <b>INTEL PACKAGING SPECIFICATIONS</b><br>Listing of Packaging types, number of leads,<br>and dimensions | 231369       | <b>No charge</b>          |

\*These prices are for the U. S. and Canada only. In Europe and other international locations, please contact your local Intel Sales Office or Distributor for literature prices.



# MICROCONTROLLER HANDBOOK

## 1986

The design on our front cover is an abstract portrayal of the control function of a microcontroller. The center sphere contains a symbolic microcontroller guiding multi-levels of remote controlled applications around the outside of the design. Microcontrollers are improving the quality and capabilities of the end product; Intel microcontrollers do more, so you can do more.





## MICROCONTROLLER HANDBOOK

Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

Intel retains the right to make changes to these specifications at any time, without notice.

Contact your local sales office to obtain the latest specifications before placing your order.

The following are trademarks of Intel Corporation and may only be used to identify Intel Products:

1986

BITBUS, COMMputer, CREDIT, Data Pipeline, GENIUS, i<sup>+</sup>, ICE, iCS, iDBP, iDIS, i<sup>2</sup>ICE, iLBX, i<sub>m</sub>, iMDDX, iMMX, Insite, Intel, int<sub>el</sub>, int<sub>el</sub>BOS, Intelelevision, int<sub>el</sub>igent Identifier, int<sub>el</sub>igent Programming, Inteltec, Intellink, iOSP, iPDS, iRMX, iSBC, iSBX, iSDM, iSXM, KEPROM, Library Manager, MCS, Megachassis, MICROMAINFRAME, MULTIBUS, MULTICHANNEL, MULTIMODULE, OpenNET, Plug-A-Bubble, PROMPT, Promware, QUEST, QueX, Ripplemode, RMX/80, RUPI, Seamless, SLD, and UPI, and the combination of ICE, iCS, iRMX, iSBC, iSBX, MCS, or UPI and a numerical suffix.

MDS is an ordering code only and is not used as a product name or trademark. MDS® is a registered trademark of Mohawk Data Sciences Corporation.

\* MULTIBUS is a patented Intel bus.

Additional copies of this manual or other Intel literature may be obtained from:

Intel Corporation  
Literature Distribution  
Mail Stop SC6-714  
3065 Bowers Avenue  
Santa Clara, CA 95051



## Table of Contents

|  |       |
|--|-------|
| <b>ALPHANUMERIC INDEX</b> . . . . .  | iii   |
| <b>MCS®-96 FAMILY</b>  |       |
| <b>CHAPTER 1</b>   |       |
| Introduction To MCS®-96 . . . . .  | 1-1   |
| <b>CHAPTER 2</b>   |       |
| Architectural Overview . . . . .   | 2-1   |
| <b>CHAPTER 3</b>   |       |
| MCS®-96 Software Design Information . . . . .  | 3-1   |
| <b>CHAPTER 4</b>   |       |
| MCS®-96 Hardware Design Information . . . . .  | 4-1   |
| <b>CHAPTER 5</b>   |       |
| MCS®-96 Data Sheets . . . . .  | 5-1   |
| <b>CHAPTER 6</b>   |       |
| MCS®-96 Article Reprint  |       |
| AP-248: Using The 8096 . . . . .   | 6-1   |
| <b>MCS®-51 FAMILY</b>  |       |
| <b>CHAPTER 7</b>   |       |
| MCS®-51 Architecture . . . . .   | 7-1   |
| <b>CHAPTER 8</b>   |       |
| MCS®-51 Programmer's Guide and Instruction Set . . . . .                                       | 8-1   |
| <b>CHAPTER 9</b>   |       |
| MCS®-51 Data Sheets  |       |
| 8031/8051 8031AH/8051AH 8032AH/8052AH 8751H/8751H-12/8751H-88 . . . . .                        | 9-1   |
| 8052AH-Basic . . . . .   | 9-16  |
| 80C51BH/80C51BH-1/80C51BH-2 80C31BH/80C31BH-1/80C31BH-2 . . . . .                              | 9-25  |
| 80C51BH/80C31BH Express . . . . .  | 9-35  |
| 8031AH/8051AH 8032AH/8052AH 8751H/8751H Express . . . . .                                      | 9-38  |
| 8752A . . . . .  | 9-40  |
| 80C252/83C252/87C252 . . . . .   | 9-43  |
| 87C51 . . . . .  | 9-47  |
| <b>CHAPTER 10</b>  |       |
| MCS®-51 Application Notes  |       |
| AP-70: Using The Intel MCS®-51 Boolean Processing Capabilities . . . . .                       | 10-1  |
| AP-252: Designing With The 80C51BH . . . . .   | 10-35 |
| <b>CHAPTER 11</b>  |       |
| MCS®-51 Article Reprints   |       |
| AR-374: Built-In Basic Interpreter Turns Controller Chip Into Versatile System Core . . . . .  | 11-1  |
| AR-409: Increased Functions In Chip Result In Lighter, Less Costly Portable Computer . . . . . | 11-9  |



## MCS®-48 FAMILY

### CHAPTER 12

|   |      |
|---|------|
| MCS®-48 Single Component System . . . . . | 12-1 |
|---|------|

### CHAPTER 13

|                                   |      |
|-----------------------------------|------|
| MCS®-48 Expanded System . . . . . | 13-1 |
|-----------------------------------|------|

### CHAPTER 14

|                                   |      |
|-----------------------------------|------|
| MCS®-48 instruction Set . . . . . | 14-1 |
|-----------------------------------|------|

### CHAPTER 15

|  |       |
|--|-------|
| MCS®-48 Data Sheets . . . . .                          | 15-1  |
| 8243 . . . . .   | 15-7  |
| 8048AH/8035AHL/8049AH/8039AHL/8050AH/8040AHL . . . . . | 15-17 |
| 8748H/8035H/8749H/8039H . . . . .                      | 15-30 |
| MCS®-48 Express . . . . .                              | 15-34 |
| 80C49-7/80C39-7 . . . . .                              | 15-34 |

## THE RUP1™ FAMILY: MICROCONTROLLER WITH ON-CHIP COMMUNICATION CONTROLLER

### CHAPTER 16

|                               |      |
|-------------------------------|------|
| The RUP1™-44 Family . . . . . | 16-1 |
|-------------------------------|------|

### CHAPTER 17

|                             |      |
|-----------------------------|------|
| 8044 Architecture . . . . . | 17-1 |
|-----------------------------|------|

### CHAPTER 18

|                                 |      |
|---------------------------------|------|
| 8044 Serial Interface . . . . . | 18-1 |
|---------------------------------|------|

### CHAPTER 19

|                                     |      |
|-------------------------------------|------|
| 8044 Application Examples . . . . . | 19-1 |
|-------------------------------------|------|

### CHAPTER 20

|                               |      |
|-------------------------------|------|
| RUP1™ Data Sheets . . . . .   | 20-1 |
| 8044AH/8344AH/8744H . . . . . | 21-1 |

### CHAPTER 21

|                                 |      |
|---------------------------------|------|
| RUP1™ Article Reprint . . . . . | 21-1 |
|---------------------------------|------|

## DESIGN CONSIDERATIONS

### CHAPTER 22

|  |       |
|--|-------|
| Application Notes . . . . .  | 22-1  |
| AP-125: Designing Microcontroller Systems<br>For Electrically Noisy Environments . . . . . | 22-23 |
| AP-155: Oscillators For Microcontrollers . . . . .   | 22-23 |

# ALPHANUMERICAL INDEX

|                                |                     |       |
|--------------------------------|---------------------|-------|
| 8031                           | Data Sheet          | 9-1   |
| 8031AH                         | Data Sheet          | 9-1   |
| 8031AH                         | Express Data Sheet  | 9-38  |
| 8032AH                         | Data Sheet          | 9-1   |
| 8032AH                         | Data Sheet          | 9-38  |
| 8035H                          | Data Sheet          | 15-7  |
| 8035AHL                        | Data Sheet          | 15-7  |
| 8039H                          | Data Sheet          | 15-7  |
| 8039AHL                        | Data Sheet          | 15-7  |
| 8040AHL                        | Data Sheet          | 15-7  |
| 8044                           | Application Example | 19-1  |
| 8044                           | Architecture        | 17-1  |
| 8044                           | Serial Interface    | 18-1  |
| 8044AH                         | Data Sheet          | 20-1  |
| 8048AH                         | Data Sheet          | 15-7  |
| 8049AH                         | Data Sheet          | 15-7  |
| 8050AH                         | Data Sheet          | 15-7  |
| 8051                           | Data Sheet          | 9-1   |
| 8051AH                         | Data Sheet          | 9-1   |
| 8051AH                         | Express Data Sheet  | 9-38  |
| 8052AH                         | Data Sheet          | 9-1   |
| 8052AH                         | Basic Data Sheet    | 9-16  |
| 8052AH                         | Express Data Sheet  | 9-38  |
| 8243                           | Data Sheet          | 15-1  |
| 8344AH                         | Data Sheet          | 20-1  |
| 8744H                          | Data Sheet          | 20-1  |
| 8748H                          | Data Sheet          | 15-17 |
| 8749H                          | Data Sheet          | 15-7  |
| 8751H                          | Data Sheet          | 9-1   |
| 8751H                          | Express Data Sheet  | 9-38  |
| 8751H-12                       | Data Sheet          | 9-1   |
| 8752A                          | Data Sheet          | 9-40  |
| 8751H-88                       | Data Sheet          | 9-1   |
| 80C31BH                        | Data Sheet          | 9-25  |
| 80C31BH                        | Express Data Sheet  | 9-35  |
| 80C31BH-1                      | Data Sheet          | 9-25  |
| 80C31BH-2                      | Data Sheet          | 9-25  |
| 80C39-7                        | Data Sheet          | 15-34 |
| 80C49-7                        | Data Sheet          | 15-34 |
| 80C51BH                        | Data Sheet          | 9-25  |
| 80C51BH                        | Express Data Sheet  | 9-35  |
| 80C51BH-1                      | Data Sheet          | 9-25  |
| 80C51BH-2                      | Data Sheet          | 9-25  |
| 80C252                         | Data Sheet          | 9-43  |
| 83C252                         | Data Sheet          | 9-43  |
| 87C51                          | Data Sheet          | 9-47  |
| 87C252                         | Data Sheet          | 9-43  |
| ADVANCED Packaging Information |                     | 24-1  |
| Design Considerations          |                     | 22-1  |



# ALPHANUMERICAL INDEX

# ALPHANUMERICAL INDEX

|   |                                 |
|---|---------------------------------|
| Design Considerations Application Notes                 | 22-2, 22-24                     |
| Design Considerations When Using CHMOS                  | 23-1                            |
| Design Considerations When Using CHMOS Article Reprints | 23-6, 23-17                     |
| MCS-48 Data Sheet                                       | 15-1, 15-7, 15-17, 15-30, 15-34 |
| MCS-48 Expanded System                                  | 13-1                            |
| MCS-48 Instruction Set                                  | 14-1                            |
| MCS-48 Single Component System                          | 12-1                            |
| MCS-51 Application Notes                                | 10-1, 10-31, 10-65              |
| MCS-51 Architecture                                     | 7-1                             |
| MCS-51 Article Reprint                                  | 11-1                            |
| MCS-51 Data Sheet                                       | 9-1, 9-16, 9-25, 9-38           |
| MCS-51 Programmer's Guide and Instruction Set           | 8-1                             |
| MCS-96 Architectural Overview                           | 2-1                             |
| MCS-96 Article Reprint                                  | 6-1                             |
| MCS-96 Data Sheet                                       | 5-1                             |
| MCS-96 Hardware Information                             | 4-1                             |
| MCS-96 Introduction                                     | 1-1                             |
| MCS-96 Software Design Information                      | 3-1                             |
| RUPI Data Sheets  | 20-1, 20-20                     |
| RUPI Article Reprints                                   | 21-1                            |
| RUPI-44 Family  | 16-1                            |

## CUSTOMER SUPPORT

### CUSTOMER SUPPORT

Customer Support is Intel's complete support service that provides Intel customers with Customer Training, Software Support and Hardware Support.

After a customer purchases any system hardware or software product, service and support become major factors in determining whether that product will continue to meet a customer's expectations. Such support requires an international support organization and a breadth of programs to meet a variety of customer needs. Intel's extensive customer support includes factory repair services as well as worldwide field service offices providing hardware repair services, software support services and customer training classes.

### HARDWARE SUPPORT

Hardware Support Services provides maintenance on Intel supported products at board and system level. Both field and factory services are offered. Services include several types of field maintenance agreements, installation and warranty services, hourly contracted services (factory return for repair) and specially negotiated support agreements for system integrators and large volume end-users having unique service requirements. For more information contact your local Intel Sales Office.

### SOFTWARE SUPPORT

Software Support Service provides maintenance on software packages via software support contracts which include subscription services, information phone support, and updates. Consulting services can be arranged for on-site assistance at the customer's location for both short-term and long-term needs. For complex products such as NDS II or I<sup>2</sup>CICE, orientation/installation packages are available through membership in Insite User's Library, where customer-submitted programs are catalogued and made available for a minimum fee to members. For more information contact your local Intel Sales Office.

### CUSTOMER TRAINING

Customer Training provides workshops at customer sites (by agreement) and on a regularly scheduled basis at Intel's facilities. Intel offers a breadth of workshops on microprocessors, operating systems and programming languages, etc. For more information on these classes contact the Training Center nearest you.

### TRAINING CENTER LOCATIONS

To obtain a complete catalog of our workshops, call the nearest Training Center in your area.

|                    |                   |                     |                 |
|--------------------|-------------------|---------------------|-----------------|
| Boston             | (617) 692-1000    | London              | (0793) 696-000  |
| Chicago            | (312) 310-5700    | Munich              | (089) 5389-1    |
| San Francisco      | (415) 940-7800    | Paris               | (01) 687-22-21  |
| Washington, D.C.   | (301) 474-2878    | Stockholm           | (468) 734-01-00 |
| Israel             | (972) 349-491-099 | Milan               | 39-2-82-44-071  |
| Tokyo              | 03-437-6611       | Benelux (Rotterdam) | (10) 21-23-77   |
| Osaka (Call Tokyo) | 03-437-6611       | Copenhagen          | (1) 198-033     |
| Toronto, Canada    | (416) 675-2105    | Hong Kong           | 5-215311-7      |



# CUSTOMER SUPPORT

## CUSTOMER SUPPORT

Customer Support is Intel's complete support service that provides Intel customers with Customer Training, Software Support and Hardware Support.

After a customer purchases any system hardware or software product, service and support become major factors in determining whether that product will continue to meet a customer's expectations. Such support requires an international support organization and a breadth of programs to meet a variety of customer needs. Intel's extensive customer support includes factory repair services as well as worldwide field service offices providing hardware repair services, software support services and customer training classes.

## HARDWARE SUPPORT

Hardware Support Services provides maintenance on Intel supported products at board and system level. Both field and factory services are offered. Services include several types of field maintenance agreements, installation and warranty services, hourly contracted services (factory return for repair) and specially negotiated support agreements for system integrators and large volume end-users having unique service requirements. For more information contact your local Intel Sales Office.

## SOFTWARE SUPPORT

Software Support Service provides maintenance on software packages via software support contracts which include subscription services, information phone support, and updates. Consulting services can be arranged for on-site assistance at the customer's location for both short-term and long-term needs. For complex products such as NDS II or PICE, on-site installation packages are available through membership in Intel's User's Library, where customer-submitted programs are cataloged and made available for a minimum fee to members. For more information contact your local Intel Sales Office.

## CUSTOMER TRAINING

Customer Training provides workshops at customer sites (by agreement) and on a regularly scheduled basis at Intel facilities. Intel offers a breadth of workshops on microprocessors, operating systems and programming languages, etc. For more information on these classes contact the Training Center nearest you.

## TRAINING CENTER LOCATIONS

To obtain a complete catalog of our workshops, call the nearest Training Center in your area.

|                    |                   |                     |                 |
|--------------------|-------------------|---------------------|-----------------|
| Boston             | (617) 692-1000    | London              | (0203) 698-000  |
| Chicago            | (312) 310-2700    | Munich              | (089) 2389-1    |
| San Francisco      | (415) 940-7800    | Paris               | (01) 687-22-21  |
| Washington, D.C.   | (301) 474-2878    | Stockholm           | (468) 734-01-00 |
| Israel             | (022) 349-401-009 | Milan               | 39-2-82-44-071  |
| Tokyo              | 03-437-6611       | Benelux (Rotterdam) | (10) 21-22-77   |
| Ozaka (Call Tokyo) | 03-437-6611       | Copenhagen          | (1) 198-033     |
| Toronto, Canada    | (416) 672-2102    | Hong Kong           | 2-212311-7      |







# CHAPTER 1 INTRODUCTION TO MCS®-96

## 1.0 MCS-96 SECTION ORGANIZATION

The MCS-96 family of microcontrollers is composed of the 8096 series of components and the 8096BH series. The 8096BH parts are fully compatible with the 8096 parts while offering enhanced functionality and the option of on-chip EPROM.

The following five chapters describe the MCS-96 family. To provide a convenient means of reference for those already familiar with the products, each chapter is a free standing module which covers one aspect of design. For those not already familiar with the 8096, reading the chapters in the order they appear will present a logical and complete presentation of the MCS-96 family of products.

Chapter 1 presents an introduction to the MCS-96 product family.

An Architectural Overview, contained in Chapter 2, provides the operational description of each of the hardware units on the chip. Information in this chapter will be of interest to anyone technically involved with an 8096 design.

The Software Design section, Chapter 3, provides information

which will primarily interest those people who will write programs to execute in the 8096.

The Hardware Design section, Chapter 4, provides the hardware engineer with all the information needed to connect external hardware to the 8096.

Chapters 3 and 4 are both written assuming the reader is familiar with the information contained in Chapter 2.

Data sheets for the MCS-96 parts are contained in Chapter 5. The first data sheet describes the 8096 series of parts. The second one describes the 8096BH series of parts. The final data sheet is for the Express series of parts, those that have been burned-in and/or tested for an extended temperature range.

## 1.1 CONTINUING MICROCONTROLLER EVOLUTION

Beginning with the introduction of the world standard 8048 (MCS®-48) Microcontroller in 1976, Intel has continued to drive the evolution of single chip microcontrollers. In 1980, Intel introduced the 8051 (MCS-51) offering

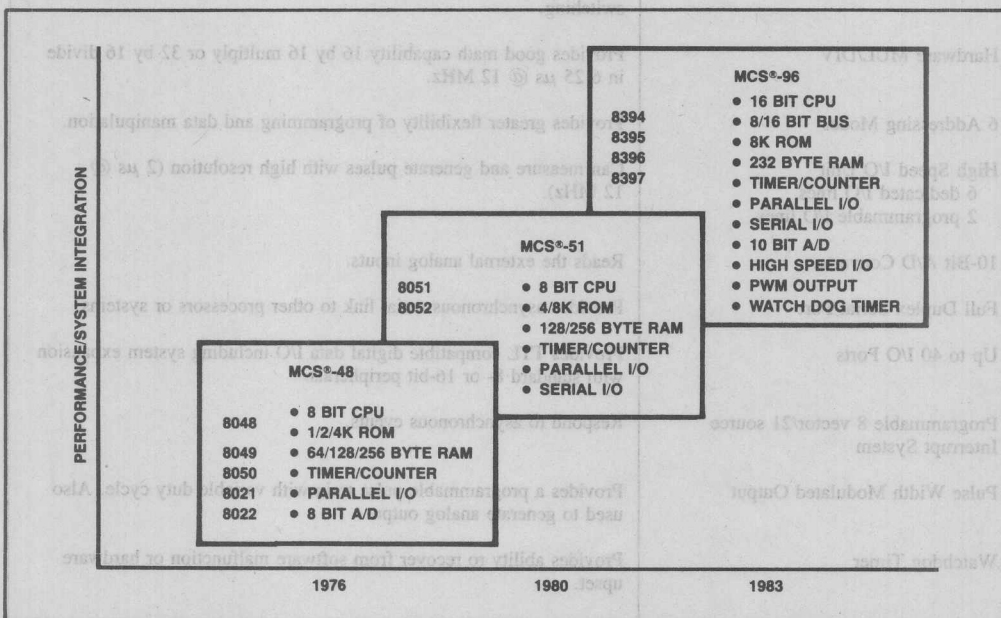


Figure 1-1. Evolution of Microcontrollers at Intel



## INTRODUCTION TO MCS®-96

performance levels significantly higher than the 8048. With the advent of the 8051, the microcontroller applications base took a marked vertical leap. These versatile chips are used in applications from keyboards and terminals to controlling automobile engines. The 8051 quickly gained the position of the second generation world standard microcontroller.

Now that the semiconductor process technologies are being pushed to new limits, it has become possible to integrate more than 100,000 transistors onto a single silicon chip. Microcontroller designers at Intel have taken today's process technology achievements and forged a new generation of single chip microcontrollers called the MCS-96. The 8096 (generic part number for MCS-96) offers the highest level of system integration ever achieved on a single chip microcontroller. It uses over 120,000 transistors to implement a high performance 16-bit CPU.

8K bytes of program memory, 232 bytes of data memory and both analog and digital types of I/O features. Figure 1-1 shows the evolution of single chip microcontroller at Intel.

### 1.2 INTRODUCTION TO THE MCS®-96

The 8096 consists of a powerful 16-bit CPU tightly coupled with program and data memory along with several I/O features all integrated onto a single piece of silicon. The CPU supports bit, byte, and word operations. 32-bit double words are also supported for a subset of the instruction set. With a 12 MHz input frequency, the 8096 can perform a 16-bit addition in 1.0  $\mu$ s and 16 x 16 multiply or 32/16 divide in 6.25  $\mu$ s.

The 8096BH allows the external bus width to be run-time configured to operate as a standard 16-bit multiplexed address/data bus or an 8088 minimum mode type bus.

Table 1-1 MCS®-96 Features and Benefits Summary

| FEATURES   | BENEFITS   |
|--|--|
| 16-Bit CPU   | Efficient machine with higher throughput.  |
| Dynamically Reconfigurable Bus   | Select 8-bit or 16-bit bus width.  |
| 8K Bytes ROM   | Large program space for more complex, larger programs.   |
| 232 Bytes RAM  | Large on-board register file for data storage and fast context switching.  |
| Hardware MUL/DIV   | Provides good math capability 16 by 16 multiply or 32 by 16 divide in 6.25 $\mu$ s @ 12 MHz.   |
| 6 Addressing Modes   | Provides greater flexibility of programming and data manipulation.   |
| High Speed I/O Unit<br>6 dedicated I/O lines<br>2 programmable I/O lines     | Can measure and generate pulses with high resolution (2 $\mu$ s @ 12 MHz).   |
| 10-Bit A/D Converter   | Reads the external analog inputs.  |
| Full Duplex Serial Port  | Provides asynchronous serial link to other processors or systems.  |
| Up to 40 I/O Ports   | Provides TTL compatible digital data I/O including system expansion with standard 8- or 16-bit peripherals.                          |
| Programmable 8 vector/21 source Interrupt System                             | Respond to asynchronous events.  |
| Pulse Width Modulated Output   | Provides a programmable pulse train with variable duty cycle. Also used to generate analog output.                                   |
| Watchdog Timer   | Provides ability to recover from software malfunction or hardware upset.   |
| 48-Pin (DIP) & 68-Pin (Plastic Leaded Chip Carrier, Pin Grid Array) Versions | Offers a variety of package types to choose from to better fit a specific application need for number of I/O lines and package size. |

Four high-speed trigger inputs are provided to record the times at which external events occur with a resolution of 2  $\mu$ s (at 12 MHz crystal frequency). Up to six high-speed pulse generator outputs are provided to trigger external events at preset times. The high speed output unit can simultaneously perform software timer functions. Up to four such 16-bit software timers can be in operation at once in addition to the two 16-bit hardware timers.

An optional on-chip A/D converter converts up to eight analog input channels into 10-bit digital values. Also provided on-chip, are a serial port, a watchdog timer, and a pulse-width modulated output signal. Table 1.1 shows the features and benefits summary for the MCS-96.

The 8096 with its 16-bit CPU and all the I/O features and interface resources on a single piece of silicon represents

**Table 1-2 MCS®-96 Broad Base of Applications**

## INDUSTRIAL

- Motor Control
- Robotics
- Discrete and Continuous Process Control
- Numerical Control
- Intelligent Transducers

## INSTRUMENTATION

- Medical Instrumentation
- Liquid and Gas Chromatographs
- Oscilloscopes

## CONSUMER

- Video Recorder
- Laser Disk Drive
- High-end Video Games

## GUIDANCE & CONTROL

- Missile Control
- Torpedo Guidance Control
- Intelligent Ammunition
- Aerospace Guidance Systems

## DATA PROCESSING

- Plotters
- Color and B&W Copiers
- Winchester Disk Drive
- Tape Drives
- Impact and Non-Impact Printers

## TELECOMMUNICATIONS

- Modems
- Intelligent Line Card Control

## AUTOMOTIVE

- Ignition Control
- Transmission Control
- Anti Skid Braking
- Emission Control

the highest level of system integration in the world of microcontrollers. It will open up new applications which had to use multiple chip solutions in the past.

## 1.3. MCS®-96 APPLICATIONS

The MCS-96 products are stand-alone high performance single chip microcontrollers designed for use in sophisticated real-time demanding applications such as industrial control, instrumentation and intelligent computer peripherals. The wide base of applications cut across all industry segments (see table 1.2). With the 16-bit CPU horsepower, high-speed math processing and high-speed I/O, the 8096 is ideal for complex motor control and axis control systems. Examples include three phase, large horsepower AC motors and robotics.

With its 10-bit A/D converter option, the device finds usage in data acquisition systems and closed-loop analog controllers. It permits considerable system integration by combining analog and digital I/O processing in the single chip.

This chip is ideally suited in the area of instrumentation products such as gas chromatographs, which combine analog processing with high speed number crunching. The same features make it a desirable component for aerospace applications like missile guidance and control.

## 1.4. MCS®-96 FAMILY DEVELOPMENT SUPPORT TOOLS

The product family is supported by a range of Intel software and hardware development tools. These tools shorten the product development cycle, thus bringing the product to the market sooner.

### 1.4.1. MCS®-96 Software Development Package

The 8096 software development package provides development system support specifically designed for the MCS-96 family of single chip microcontrollers. The package consists of a symbolic macro assembler ASM-96, Linker/Relocator RL-96 and the librarian LIB-96. Among the high level language, PLM-96 is offered along with a floating point math package. A real-time executive software package, the iDCX-96 is also available. Additional high level languages are being developed for the MCS-96 product family.

### 1.4.2. ASM-96 MACRO Assembler

The 8096 macro assembler translates the symbolic assembly language instructions into the machine executable object code. ASM-96 enables the programmer to write the program in a modular fashion. The modular programs divide a rather complex program into smaller functional units, that are easier to code, to debug, and to change. The separate modules can then be linked and located into one program module using the RL-96 utility. This utility

combines the selected input object modules into a single output object module. It also allocates memory to input segments and binds the relocatable addresses to absolute addresses. It then produces a print file that consists of a link summary, a symbol table listing and an intermediate cross-reference listing. LIB-96, another utility helps to create, modify, and examine library files. The ASM-96 runs on Intellec Series III or IV.

#### 1.4.3. PL/M-96

The PL/M-96 compiler translates the PL/M-96 language into 8096 relocatable object modules. This allows improved programmer productivity and application reliability. This high level language has been efficiently designed to map into the machine architecture, so as not to trade off higher programmer productivity with inefficient code. Since the language and the compiler are optimized for the 8096 and its application environment, developing software with PL/M-96 is a 'low-risk' project.

#### 1.4.4. iDCX-96

The iDCX-96 is an executive software package useful in multi-tasking environments. Up to 16 user tasks can be handled. The iDCX-96 is ideal for the customer who must quickly develop software for a multi-tasking, real-time environment.

#### 1.4.5. Hardware Development Support: VLSICE-96

The VLSICE-96 is based on Intel's state-of-the-art bond-out technology. It permits full access to the internal bus and control timing cycles for true real-time emulation. The VLSICE-96 is controlled by either an Intellec series III/IV or an IBM PC/XT/AT or compatible over a serial link.

The VLSICE-96 provides total development support for MCS-96 microcontroller designs. It supports full speed real time emulation. A comprehensive break/trace capability allows for the specification of complex, multi-level events.

#### 1.4.6. Hardware Development Support: iSBE-96

The iSBE-96 is a hardware execution and debug tool for the MCS-96 products. It consists of a monitor/debugger resident in an 8096 system. This development system interfaces with the user's 8096 system via two ribbon cables, one for the 8096 I/O ports, and the other for the memory bus. The iSBE-96 is controlled by an Intellec Series III, IBM PC or compatible, or other computer system over a serial link. Power for the iSBE-96 can be supplied by

plugging it into the MULTIBUS® card slot, or by an external power supply. The iSBE-96 is contained on one standard MULTIBUS board.

The iSBE-96 provides the most often used features for real-time hardware emulation. The user can display and modify memory, set up break points, execute with or without breakpoints and change the memory map. In addition, the user can single step through the system program.

#### 1.4.7. MCS®-96 Workshop

The workshop provides the design engineer or system designer hands-on experience with the MCS-96 family of products. The course includes an explanation of the Intel 8096 architecture, system timing, input/output design. The lab sessions allow the attendees to gain in-depth knowledge of the MCS-96 product family and support tools.

#### 1.4.8. Insite™ Library

The Intel Insite Library contains several application programs. A very useful program contained in the Insite is SIM-96, the software simulator for 8096. It allows software simulations of user's system. The simulator provides the ability to set breakpoints, examine and modify memory, disassemble the object code and single step through the code.

### 1.5. MCS®-96 FAMILY OF PRODUCTS

Although 8096 is the generic part number often used for the MCS-96 products throughout this manual, the product family consists of eight configurations with eight part numbers including the 8096. This wide variety of products is offered to best meet user's application requirements in terms of number of I/O's and package size. The options include on-board 8K bytes of mask programmed memory, 10-bit A/D converter, and 48 or 68 pin package type.

The 48-pin components are similar to the 68-pin versions except that the following pins are not available:

|         |  |
|---------|--|
| Port 0  | 4 of 8 analog/digital input pins   |
| Port 1  | 8 general purpose I/O pins   |
| Port 2  | 4 of 8 general/special function pins<br>(the special functions are available through other pins) |
| Control | CLKOUT, INSTRUCTION, NMI, TEST<br>(BUSWIDTH on 8096BH)   |

Table 1-3 summarizes all the current products in the MCS®-96 product family.



## INTRODUCTION TO MCS®-96

---

**Table 1-3 MCS®-96 Family of Products**

| OPTIONS                         |         | 68-PIN | 48-PIN |
|---------------------------------|---------|--------|--------|
| DIGITAL<br>I/O                  | ROMLESS | 8096   | 8094   |
|                                 | ROM     | 8396   | 8394   |
|                                 | EPROM   | *8796  | 8794   |
| ANALOG<br>AND<br>DIGITAL<br>I/O | ROMLESS | 8097   | 8095   |
|                                 | ROM     | 8397   | 8395   |
|                                 | EPROM   | *8797  | 8795   |

The 48 pin version is available in a DIP (dual inline) package.

The 68 pin version comes in two packages, the Plastic Leaded Chip Carrier and the Pin Grid Array.

\*The 68-pin version of the 879x is available in a Leaded Chip Carrier and a ceramic Pin Grid Array. The L.C.C. is socket foot print compatible with PLCC.









## CHAPTER 2 ARCHITECTURAL OVERVIEW

### 2.0. INTRODUCTION

The 8096 can be separated into several sections for the purpose of describing its operation. There is a CPU, a programmable High Speed I/O Unit, an analog to digital converter, a serial port, and a Pulse Width Modulated (PWM) output for digital to analog conversion. In addition to these functional units, there are some sections which support overall operation of the chip such as the clock generator and the back-bias generator. The CPU and the programmable I/O make the 8096 very different from any other microcontroller; let us first examine the CPU.

### 2.1. CPU OPERATION

The major components of the CPU on the 8096 are the Register File and the RALU. Communication with the outside world is done through either the Special Function Registers (SFRs) or the Memory Controller. The RALU (Register/Arithmetic Logic Unit) does not use an accumulator, it operates directly on the 256-byte register space made up of the Register File and the SFRs. Efficient I/O

operations are possible by directly controlling the I/O through the SFRs. The main benefits of this structure are the ability to quickly change context, the absence of accumulator bottleneck, and fast throughput and I/O times.

#### 2.1.1. CPU Buses

A "Control Unit" and two buses connect the Register File and RALU. Figure 2-1 shows the CPU with its major bus connections. The two buses are the "A-Bus" which is 8-bits wide, and the "D-Bus" which is 16-bits wide. The D-Bus transfers data only between the RALU and the Register File or Special Function Registers (SFRs). The A-Bus is used as the address bus for the above transfers or as a multiplexed address/data bus connecting to the "Memory Controller". Any accesses of either the internal ROM or external memory are done through the Memory Controller.

Within the memory controller is a slave program counter (Slave PC) which keeps track of the PC in the CPU. By having most program fetches from memory referenced to

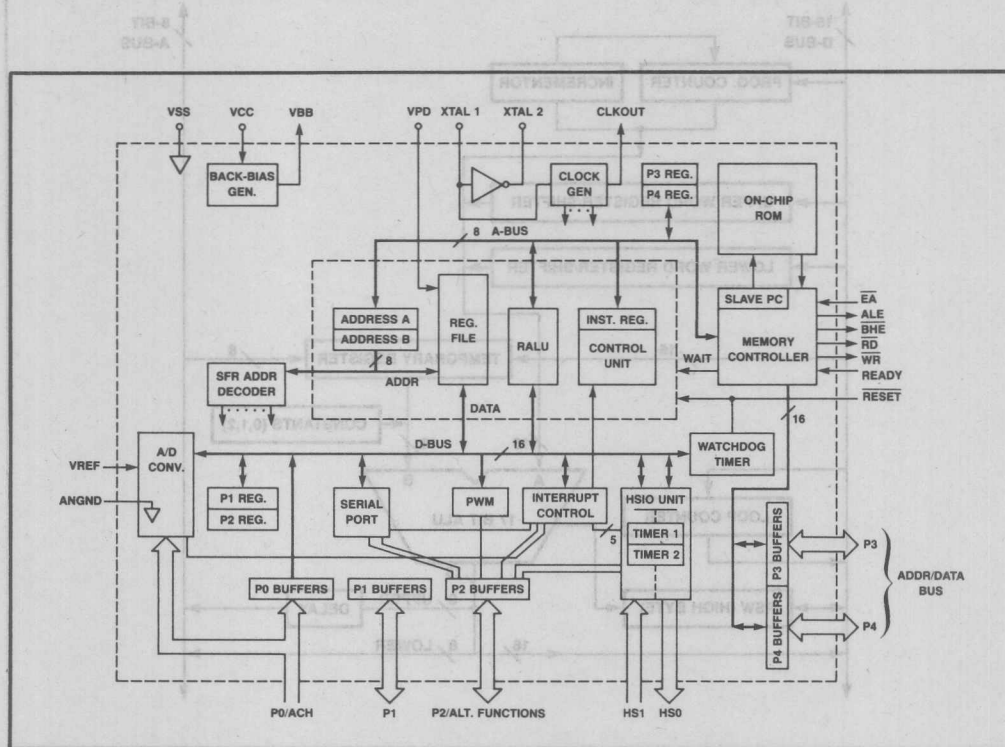


Figure 2-1. Block Diagram (For simplicity, lines connecting port registers to port buffers are not shown.)

## ARCHITECTURAL OVERVIEW

the slave PC, the processor saves time as addresses seldom have to be sent to the memory controller. If the address jumps sequence then the slave PC is loaded with a new value and processing continues. Data fetches from memory are also done through the memory controller, but the slave PC is bypassed for this operation.

### 2.1.2. CPU Register File

The Register File contains 232 bytes of RAM which can be accessed as bytes, words, or double-words. Since each of these locations can be used by the RALU, there are essentially 232 "accumulators". The first word in the Register File is reserved for use as the stack pointer so it can not be used for data when stack manipulations are taking place. Addresses for accessing the Register File and SFRs are temporarily stored in two 8-bit address registers by the CPU hardware.

### 2.1.3. RALU Control

Instructions to the RALU are taken from the A-Bus and stored temporarily in the instruction register. The Control

Unit decodes the instructions and generates the correct sequence of signals to have the RALU perform the desired function. Figure 2-1 shows the instruction register and the control unit.

### 2.1.4. RALU

Most calculations performed by the 8096 take place in the RALU. The RALU, shown in Figure 2-2, contains a 17-bit ALU, the Program Status Word (PSW), the Program Counter (PC), a loop counter, and three temporary registers. All of the registers are 16-bits or 17-bits (16 + sign extension) wide. Some of the registers have the ability to perform simple operations to off-load the ALU.

A separate incrementer is used for the PC; however, jumps must be handled through the ALU. Two of the temporary registers have their own shift logic. These registers are used for the operations which require logical shifts, including Normalize, Multiply, and Divide. The "Lower Word" register is used only when double-word quantities are being shifted, the "Upper Word" register is used

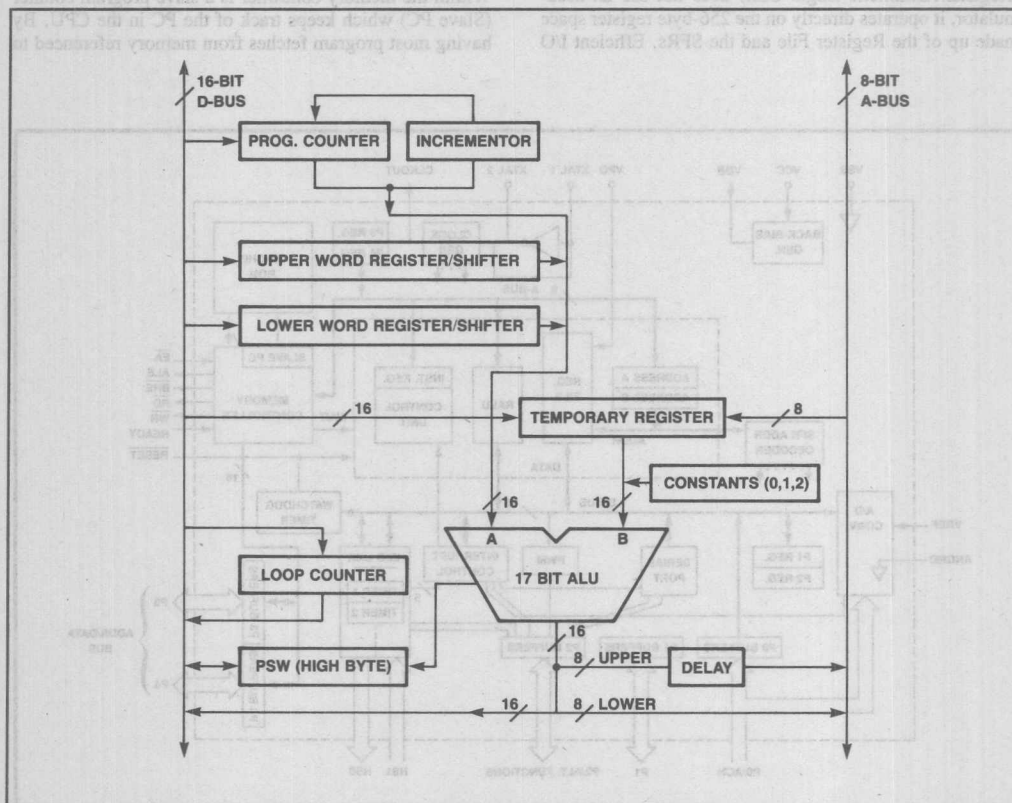


Figure 2-2. RALU Block Diagram



whenever a shift is performed or as a temporary register for many instructions. Repetitive shifts are counted by the 5-bit "Loop Counter".

A temporary register is used to store the second operand of two operand instructions. This includes the multiplier during multiplications and the divisor during divisions. To perform subtractions, the output of this register can be complemented before being placed into the "B" input of the ALU.

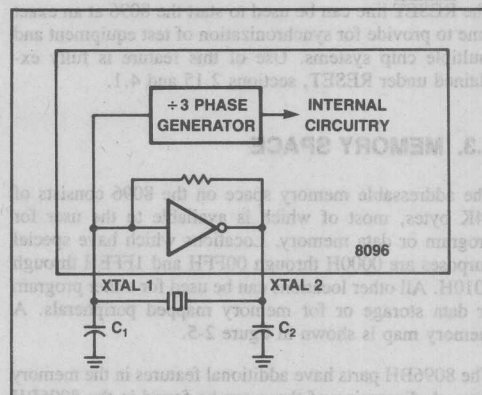
The DELAY shown in Figure 2-2 is used to convert the 16-bit bus into an 8-bit bus. This is required as all addresses and instructions are carried on the 8-bit A bus. Several constants, such as 0, 1 and 2 are stored in the RALU for use in speeding up certain calculations. These come in handy when the RALU needs to make a 2's complement number or perform an increment or decrement instruction.

## 2.2. BASIC TIMING

The 8096 requires an input clock frequency of between 6.0 MHz and 12 MHz to function. This frequency can be applied directly to XTAL1. Alternatively, since XTAL1 and XTAL2 are inputs and outputs of an inverter, it is also possible to use a crystal to generate the clock. A block diagram of the oscillator section is shown in Figure 2-3. Details of the circuit and suggestions for its use can be found in section 4.1.

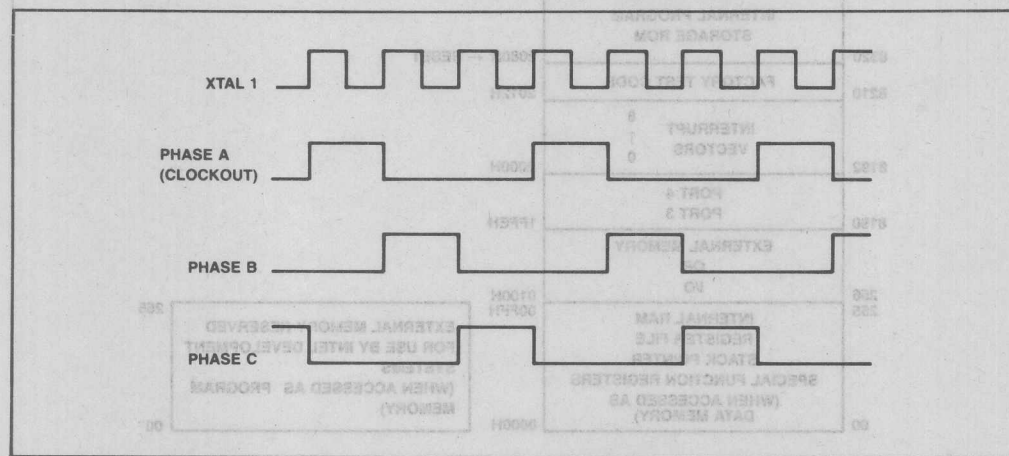
### 2.2.1. Internal Timings

The crystal or external oscillator frequency is divided by 3 to generate the three internal timing phases as shown in Figure 2-4. Each of the internal phases repeat every 3



### Figure 2-3. Block Diagram of Oscillator

oscillator periods: 3 oscillator periods are referred to as one "state time", the basic time measurement for 8096 operations. Most internal operations are synchronized to either Phase A, B or C, each of which have a 33% duty cycle. Phase A is represented externally by CLKOUT, a signal available on the 68-pin part. Phases B and C are not available externally. The relationships of XTAL1, CLKOUT, and Phases A, B, and C are shown in Figure 2-4. It should be noted that propagation delays have not been taken into account in this diagram. Details on these and other timing relationships can be found in sections 4.1, 4.4 and 4.6.



**Figure 2-4. Internal Timings Relative to XTAL 1**

The **RESET** line can be used to start the 8096 at an exact time to provide for synchronization of test equipment and multiple chip systems. Use of this feature is fully explained under **RESET**, sections 2.15 and 4.1.

## 2.3. MEMORY SPACE

The addressable memory space on the 8096 consists of 64K bytes, most of which is available to the user for program or data memory. Locations which have special purposes are 0000H through 00FFH and 1FFEH through 2010H. All other locations can be used for either program or data storage or for memory mapped peripherals. A memory map is shown in figure 2-5.

The 8096BH parts have additional features in the memory map. A discussion of these can be found in the 8096BH data sheet in chapter 5.

### 2.3.1. Register File

Locations 00H through 0FFH contain the Register File and SFRs. Complete information on this section of memory space can be found in section 2.4. No code can be executed from this internal RAM section. If an attempt to execute instructions from locations 000H through 0FFH is made, the instructions will be fetched from external memory. This section of external memory is reserved for use by Intel development tools. Execution of a nonmaskable interrupt (NMI) will force a call to external location

0000H, therefore, the NMI instruction is also reserved for Intel development tools.

### 2.3.2. Reserved Memory Spaces

Locations 1FFEH and 1FFFH are reserved for Ports 3 and 4 respectively. This is to allow easy reconstruction of these ports if external memory is used in the system. An example of reconstructing the I/O ports is given in section 4.6.7. If ports 3 and 4 are not going to be reconstructed then these locations can be treated as any other external memory location.

The 9 interrupt vectors are stored in locations 2000H through 2011H. The 9th vector is used by Intel development systems, as explained in section 2.5. Internal locations 2012H through 207FH are reserved for Intel's factory test code. To ensure compatibility with future parts external locations 2012H through 207FH must contain the hex value FFH.

Resetting the 8096 causes instructions to be fetched starting from location 2080H. This location was chosen to allow a system to have up to 8K of RAM continuous with the register file. Further information on reset can be found in section 2.15.

### 2.3.3. Internal ROM

When a ROM part is ordered, the internal memory locations 2080H through 3FFFH are user specified as are the interrupt vectors in locations 2000H through 2011H.

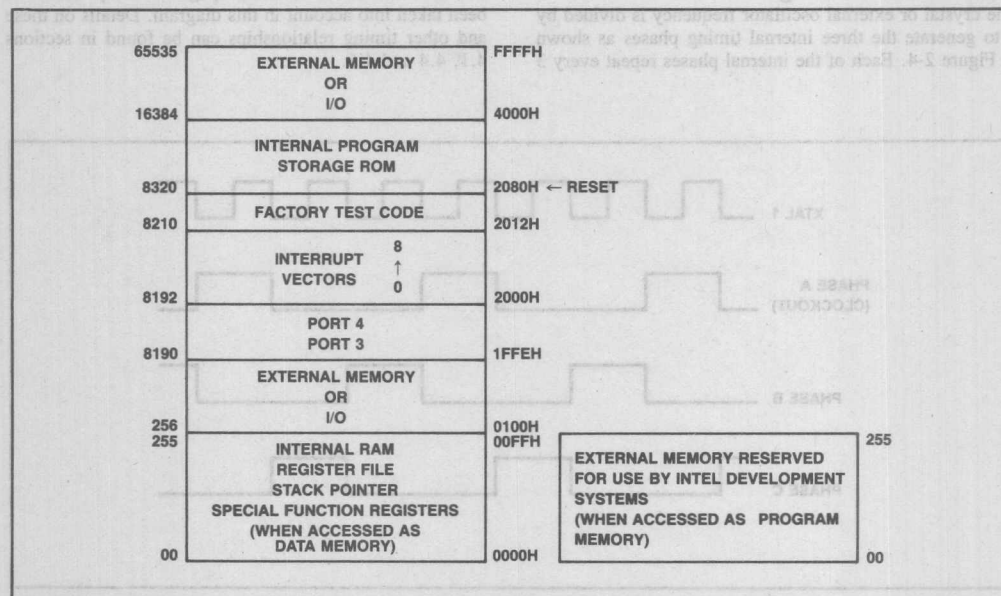


Figure 2-5. Memory Map

Instruction and data fetches from the internal ROM occur only if the part has a ROM, EA is tied high, and the address is between 2000H and 3FFFH. At all other times data is accessed from either the internal RAM space or external memory and instructions are fetched from external memory.

## 2.3.4. Memory Controller

The RALU talks to the memory (except for the locations in the register file and SFR space) through the memory controller which is connected to the RALU by the A-bus and several control lines. Since the A-bus is eight bits wide, the memory controller uses a Slave Program Counter to avoid having to always get the instruction location from the RALU. This slave PC is incremented after each fetch. When a jump or call occurs, the slave PC must be loaded from the A-bus before instruction fetches can continue.

In addition to holding a slave PC, the memory controller contains a 3 byte queue to help speed execution. This queue is transparent to the RALU and to the user unless wait states are forced during external bus cycles. The instruction execution times shown in Tables 3-3 and 3-4 show the normal execution times with no wait states added. Reloading the slave PC and fetching the first byte of the new instruction stream takes 4 state times. This

is reflected in the jump taken/not-taken times shown in Table 3-4.

## 2.3.5. System Bus

The 8096BH data sheet, in chapter 5, has additional information on the system bus and control lines of the 8096BH.

External memory is addressed through lines AD0 through AD15 which form a 16-bit multiplexed (address/data) data bus. These lines share pins with I/O ports 3 and 4. The falling edge of the Address Latch Enable (ALE) line is used to provide a clock to a transparent latch (74LS373) in order to demultiplex the bus. A typical circuit and the required timings are shown in section 4.6. Since the 8096's external memory can be addressed as either bytes or words, the decoding is controlled with two lines, Bus High Enable ( $\overline{\text{BHE}}$ ) and Address/Data Line 0 (AD0). The  $\overline{\text{BHE}}$  line must be transparently latched, just as the addresses are.

To avoid confusion during the explanation of the memory system it is reasonable to give names to the demultiplexed address/data signals. The address signals will be called MA0 through MA15 (Memory Address), and the data signals will be called MD0 through MD15 (Memory Data).

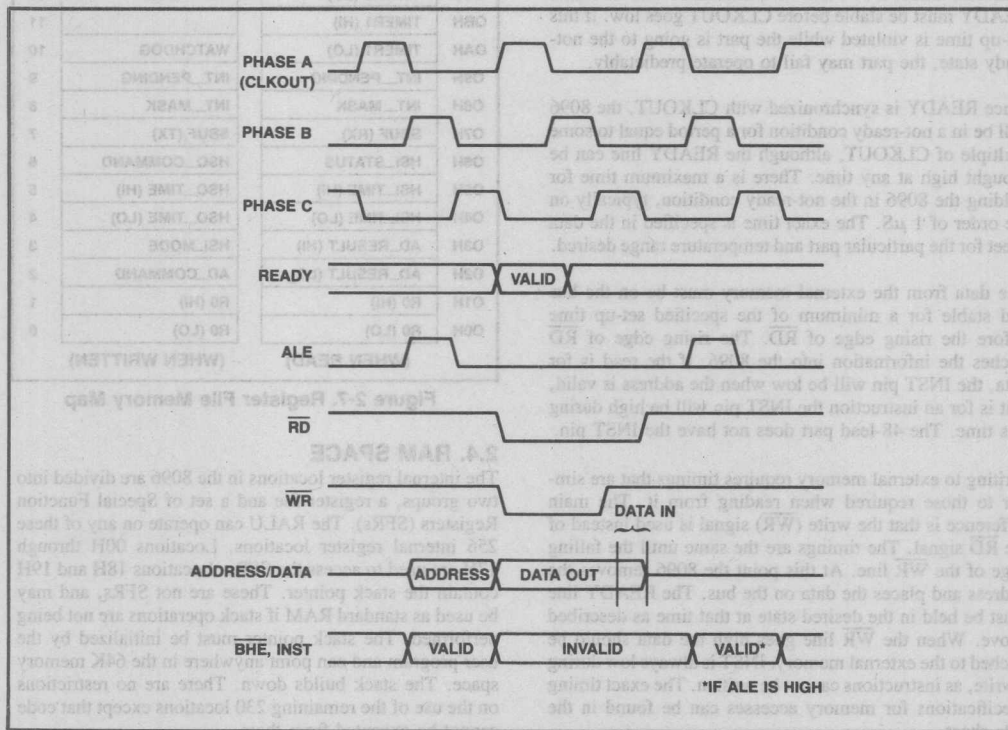


Figure 2-6. External Memory Timings

When  $\overline{BHE}$  is active (low), the memory connected to the high byte of the data bus should be selected. When  $MA0$  is low the memory connected to the low byte of the data bus should be selected. In this way accesses to a 16-bit wide memory can be to the low (even) byte only ( $MA0=0$ ,  $BHE=1$ ), to the high (odd) byte only ( $MA0=1$ ,  $BHE=0$ ), or to both bytes ( $MA0=0$ ,  $BHE=0$ ). When a memory block is being used only for reads,  $\overline{BHE}$  and  $MA0$  need not be decoded.

Figure 2-6 shows the idealized waveforms related to the following description of external memory manipulations. For exact timing specifications please refer to the latest data sheet. When an external memory fetch begins, the address latch enable (ALE) line rises, the address is put on  $AD0-AD15$  and  $BHE$  is set to the required state. ALE then falls, the address is taken off the pins, and the  $\overline{RD}$  (Read) signal goes low. The  $READY$  line can be pulled low to hold the processor in this condition for a few extra state times.

## 2.3.6. Bus Control Lines

The  $READY$  line can be used to hold the processor in the above condition in order to allow access to slow memories or for DMA purposes. Sampling of the  $READY$  line occurs internally during Phase A, which is the signal that generates  $CLKOUT$ . There is a minimum time in which  $READY$  must be stable before  $CLKOUT$  goes low. If this set-up time is violated while the part is going to the not-ready state, the part may fail to operate predictably.

Since  $READY$  is synchronized with  $CLKOUT$ , the 8096 will be in a not-ready condition for a period equal to some multiple of  $CLKOUT$ , although the  $READY$  line can be brought high at any time. There is a maximum time for holding the 8096 in the not-ready condition, typically on the order of 1  $\mu S$ . The exact time is specified in the data sheet for the particular part and temperature range desired.

The data from the external memory must be on the bus and stable for a minimum of the specified set-up time before the rising edge of  $\overline{RD}$ . The rising edge of  $\overline{RD}$  latches the information into the 8096. If the read is for data, the  $INST$  pin will be low when the address is valid, if it is for an instruction the  $INST$  pin will be high during this time. The 48-lead part does not have the  $INST$  pin.

Writing to external memory requires timings that are similar to those required when reading from it. The main difference is that the write ( $\overline{WR}$ ) signal is used instead of the  $\overline{RD}$  signal. The timings are the same until the falling edge of the  $\overline{WR}$  line. At this point the 8096 removes the address and places the data on the bus. The  $READY$  line must be held in the desired state at that time as described above. When the  $\overline{WR}$  line goes high the data should be latched to the external memory.  $INST$  is always low during a write, as instructions cannot be written. The exact timing specifications for memory accesses can be found in the data sheet.

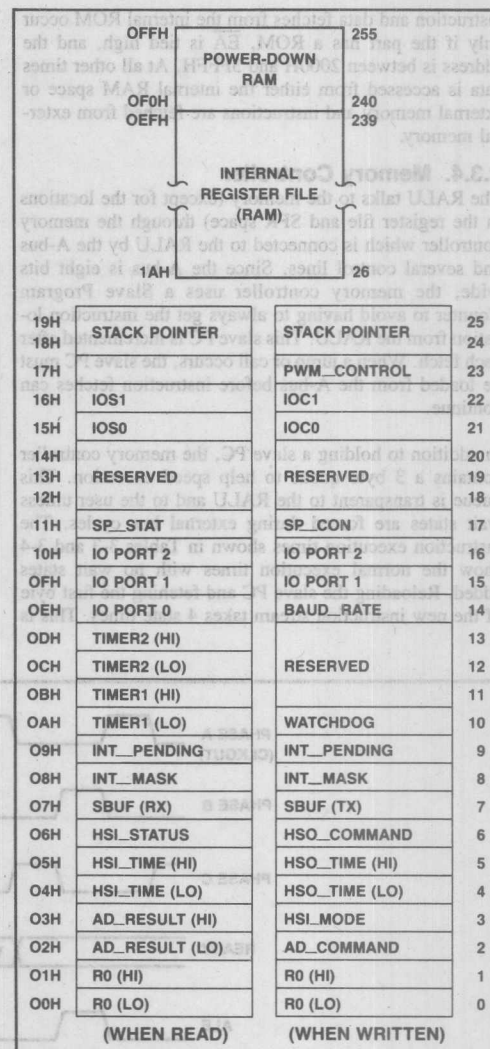


Figure 2-7. Register File Memory Map

## 2.4. RAM SPACE

The internal register locations in the 8096 are divided into two groups, a register file and a set of Special Function Registers (SFRs). The RALU can operate on any of these 256 internal register locations. Locations 00H through 17H are used to access the SFRs. Locations 18H and 19H contain the stack pointer. These are not SFRs, and may be used as standard RAM if stack operations are not being performed. The stack pointer must be initialized by the user program and can point anywhere in the 64K memory space. The stack builds down. There are no restrictions on the use of the remaining 230 locations except that code cannot be executed from them.



## 2.4.1. Special Function Registers

All of the I/O on the 8096 is controlled through the SFRs. Many of these registers serve two functions; one if they

are read from, the other if they are written to. Figure 2-7 shows the locations and names of these registers. A summary of the capabilities of each of these registers is shown

| Register      | Description  | Chapter         |
|---------------|--|-----------------|
| R0            | Zero Register — Always reads as a zero, useful for a base when indexing and as a constant for calculations and compares.   | 3.2.7           |
| AD _ RESULT   | A/D Result Hi/Low — Low and high order Results of the A/D converter (byte read only)                                       | 2.9.3           |
| AD _ COMMAND  | A/D Command Register — Controls the A/D  | 2.9.2           |
| HSI _ MODE    | HSI Mode Register — Sets the mode of the High Speed Input unit.  | 2.7.1           |
| HSI _ TIME    | HSI Time Hi/Lo — Contains the time at which the High Speed Input unit was triggered. (word read only)                      | 2.7.4           |
| HSO _ TIME    | HSO Time Hi/Lo — Sets the time for the High Speed Output to execute the command in the Command Register. (word write only) | 2.8.3           |
| HSO _ COMMAND | HSO Command Register — Determines what will happen at the time loaded into the HSO Time registers.                         | 2.8.2           |
| HSI _ STATUS  | HSI Status Registers — Indicates which HSI pins were detected at the time in the HSI Time registers.                       | 2.7.4           |
| SBUF (TX)     | Transmit buffer for the serial port, holds contents to be outputed.  | 2.11            |
| SBUF (RX)     | Receive buffer for the serial port, holds the byte just received by the serial port.                                       | 2.11            |
| INT _ MASK    | Interrupt Mask Register — Enables or disables the individual interrupts.   | 2.5.2<br>3.6.2  |
| INT _ PENDING | Interrupt Pending Register — Indicates when an interrupt signal has occurred on one of the sources.                        | 2.5.2<br>3.6.2  |
| WATCHDOG      | Watchdog Timer Register — Written to periodically to hold off automatic reset every 64K state times.                       | 2.14            |
| TIMER1        | Timer 1 Hi/Lo — Timer 1 high and low bytes. (word read only)   | 2.6.1<br>2.7-8  |
| TIMER2        | Timer 2 Hi/Lo — Timer 2 high and low bytes. (word read only)   | 2.6.2<br>2.7-8  |
| IOPORT0       | Port 0 Register — Levels on pins of port 0.  | 2.12.1          |
| BAUD _ RATE   | Register which contains the baud rate, this register is loaded sequentially.   | 2.11.4          |
| IOPORT1       | Port 1 Register — Used to read or write to Port 1.   | 2.12.2          |
| IOPORT2       | Port 2 Register — Used to read or write to Port 2.   | 2.12.3          |
| SP _ STAT     | Serial Port Status — Indicates the status of the serial port.  | 2.11.3          |
| SP _ CON      | Serial port control — Used to set the mode of the serial port.   | 2.11.1          |
| IOS0          | I/O Status Register 0 — Contains information on the HSO status.  | 2.13.4          |
| IOS1          | I/O Status Register 1 — Contains information on the status of the timers and of the HSI.                                   | 2.13.5<br>3.7.2 |
| IOC0          | I/O Control Register 0 — Controls alternate functions of HSI pins, Timer 2 reset sources and Timer 2 clock sources.        | 2.13.2          |
| IOC1          | I/O Control Register 1 — Controls alternate functions of Port 2 pins, timer interrupts and HSI interrupts.                 | 2.13.3          |
| PWM _ CONTROL | Pulse Width Modulation Control Register — Sets the duration of the PWM pulse.  | 2.10<br>4.3.2   |

Figure 2-8. SFR Summary

in Figure 2-8, with complete descriptions reserved for later chapters. Note that these registers can be accessed only as bytes unless otherwise indicated.

Within the SFR space are several registers labeled as "RESERVED". These registers are reserved for future expansion or test purposes. Reads or writes of these registers may produce unexpected results. For example, writing to location 000CH will set both timers to 0FFFXH, this feature is for use in testing the part and should not be used in programs.

## 2.4.2. Power Down

The upper 16 RAM locations (0F0H through 0FFH) receive their power from the VPD pin. If it is desired to keep the memory in these locations alive during a power down situation, one need only keep voltage on the VPD pin. The current required to keep the RAM alive is approximately 1 milliamp (refer to the data sheet for the exact specification). Both VCC and VPD must have power applied for normal operation.

To place the 8096 into a power down mode, the RESET pin is pulled low. Two state times later the part will be in reset. This is necessary to prevent the part from writing into RAM as the power goes down. The power may now be removed from the VCC pin, the VPD pin must remain within specifications. The 8096 can remain in this state for any amount of time and the 16 RAM bytes will retain their values.

To bring the 8096 out of power down, RESET is held low while VCC is applied. Two state times after the oscillator and the back bias generator have stabilized (~1 millisecond), the RESET pin can be pulled high. The 8096 will begin to execute code at location 02080H 10 state times after RESET is pulled high. Figure 2-9 shows a timing diagram of the power down sequence. To ensure that the 2 state time minimum reset time (synchronous with CLKOUT) is met, it is recommended that 10 XTAL1

cycles be used. Suggestions for actual hardware connections are given in section 4.1. Reset is discussed in section 2.15.

## 2.5. INTERRUPT STRUCTURE

### 2.5.1. Interrupt Sources

Eight interrupt sources are available on the 8096. When enabled, an interrupt occurring on any of these sources will force a call to the location stored in the vector location for that source. The interrupt sources and their respective vector locations are listed in Figure 2-10. In addition to the 8 standard interrupts, there is a TRAP instruction which acts as a software generated interrupt. This instruction is not currently supported by the MCS-96 Assembler and is reserved for use by Intel development systems. Many of the interrupt sources can be activated by several methods, Figure 2-11 shows all of the possible sources for interrupts.

| Source                  | Vector Location |            | Priority       |
|-------------------------|-----------------|------------|----------------|
|                         | (High Byte)     | (Low Byte) |                |
| Software                | 2011H           | 2010H      | Not Applicable |
| Extint                  | 200FH           | 200EH      | 7 (Highest)    |
| Serial Port             | 200DH           | 200CH      | 6              |
| Software Timers         | 200BH           | 200AH      | 5              |
| HSI.0                   | 2009H           | 2008H      | 4              |
| High Speed Outputs      | 2007H           | 2006H      | 3              |
| HSI Data Available      | 2005H           | 2004H      | 2              |
| A/D Conversion Complete | 2003H           | 2002H      | 1              |
| Timer Overflow          | 2001H           | 2000H      | 0 (Lowest)     |

Figure 2-10. Interrupt Vector Locations

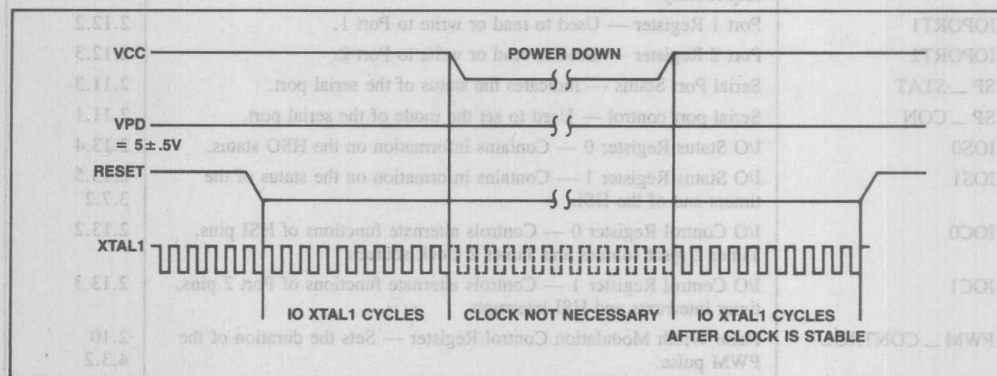


Figure 2-9. Power Down Timing

## 2.5.2. Interrupt Control

A block diagram of the interrupt system is shown in Figure 2-12. Each of the interrupt sources is tested for a 0 to 1 transition. If this transition occurs, the corresponding bit in the Interrupt Pending Register, located at 0009H, is set. The bit is cleared when the vector is taken to the interrupt routine. Since this register can be written to, it is possible to generate software interrupts by setting bits within the register, or remove pending interrupts by clearing the bits in this register. The pending register can be set even if the interrupt is disabled.

Caution must be used when writing to the pending register to clear interrupts. If the interrupt has already been acknowledged when the bit is cleared, a 4 state time "partial" interrupt cycle will occur. This is because the 8096 will have to fetch the next instruction of the normal instruction flow, instead of proceeding with the interrupt processing as it was going to. The effect on the program will be essentially that of an extra NOP. This can be prevented by clearing the bits using a 2 operand immediate logical, as the 8096 holds off acknowledging interrupts during these "read/modify/write" instructions.

Enabling and disabling of individual interrupts is done through the Interrupt Mask Register, located at 0008H. If the bit in the mask register is a 1 then the interrupt is enabled, otherwise it is disabled. Even if an interrupt is masked it may still become pending. It may, therefore, be desirable to clear the pending bit before unmasking an interrupt.

The Interrupt Mask Register is also the low byte of the PSW. All of the interrupts may be enabled and disabled simultaneously by using the "EI" (Enable Interrupt) and "DI" (Disable Interrupt) instructions. EI and DI set and clear PSW.9, the interrupt enable bit, they do not effect the contents of the mask register.

## 2.5.3. Interrupt Priority Programming

The priority encoder looks at all of the interrupts which are *both pending and enabled*, and selects the one with the highest priority. The priorities are shown in Figure 2-10 (7 is highest, 0 is lowest.) The interrupt generator then forces a call to the location in the indicated vector location. This location would be the starting location of the Interrupt Service Routine (ISR).

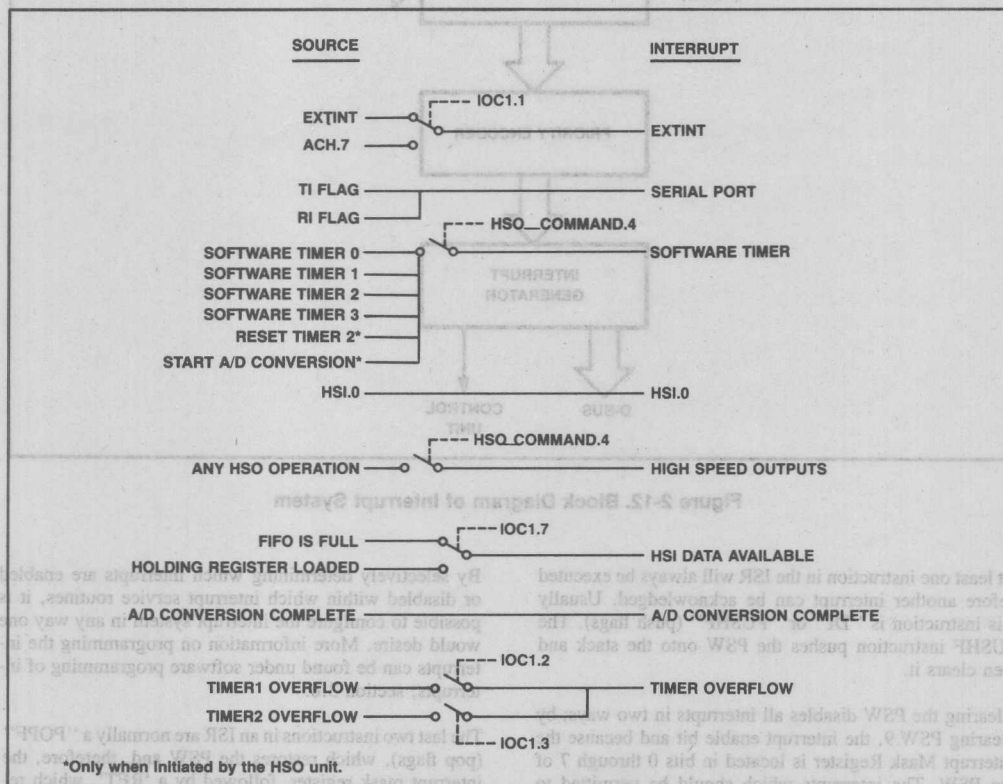


Figure 2-11. All Possible Interrupt Sources

## ARCHITECTURAL OVERVIEW

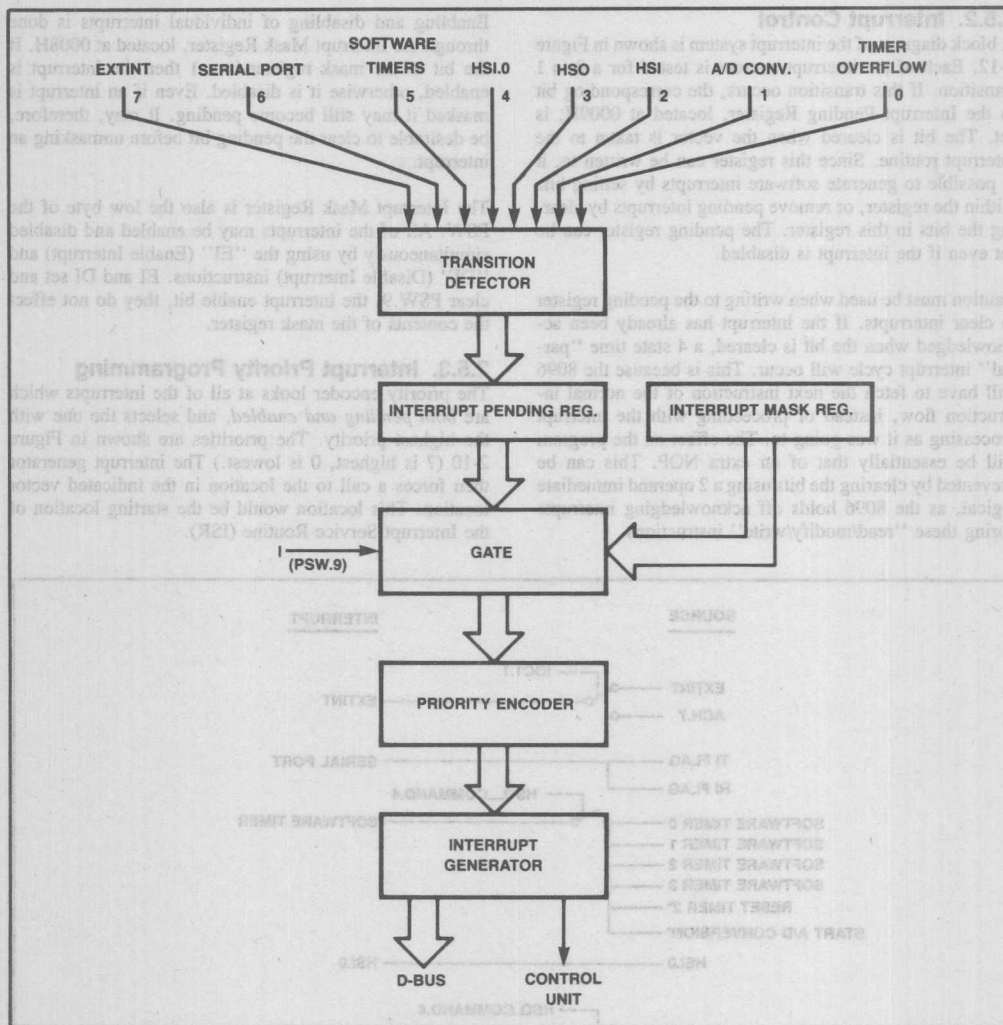


Figure 2-12. Block Diagram of Interrupt System

At least one instruction in the ISR will always be executed before another interrupt can be acknowledged. Usually this instruction is "DI" or "PUSHF" (push flags). The PUSHF instruction pushes the PSW onto the stack and then clears it.

Clearing the PSW disables all interrupts in two ways; by clearing PSW.9, the interrupt enable bit and because the Interrupt Mask Register is located in bits 0 through 7 of the PSW. The interrupts which should be permitted to interrupt this ISR can then be set in the mask register and an "EI" instruction executed.

By selectively determining which interrupts are enabled or disabled within which interrupt service routines, it is possible to configure the interrupt system in any way one would desire. More information on programming the interrupts can be found under software programming of interrupts, section 3.6.

The last two instructions in an ISR are normally a "POPF" (pop flags), which restores the PSW and, therefore, the interrupt mask register, followed by a "RET", which restores the Program Counter. Execution will then continue from the point at which the call was forced.



## 2.5.4. Interrupt Timing

Interrupts are not always acknowledged immediately. If the interrupt signal does not occur prior to 4 state-times before the end of an instruction, the interrupt will not be acknowledged until after the next instruction has been executed. This is because an instruction is fetched and prepared for execution a few state times before it is actually executed.

There are 6 instructions which always inhibit interrupts from being acknowledged until after the next instruction has been executed. These instructions are:

|             |   |
|-------------|---|
| EI, DI      | — Enable and Disable Interrupts   |
| POPF, PUSHF | — Pop and Push Flags  |
| SIGND       | — Prefix to perform signed multiply and divide (Note that this is not an ASM-96 Mnemonic, but is used for signed multiply and divide) |
| TRAP        | — Software interrupt  |

When an interrupt is acknowledged, a call is forced to the location indicated by the specified interrupt vector. This call occurs after the completion of the instruction in process, except as noted above. The procedure of getting the vector and forcing the call requires 21 state times. If the stack is in external RAM an additional 3 state times are required.

The maximum number of state times required from the time an interrupt is generated (not acknowledged) until the 8096 begins executing code at the desired location is the time of the longest instruction, NORML (Normalize — 43 state times), plus the 4 state times prior to the end of the previous instruction, plus the response time (21 to 24 state times). Therefore, the maximum response time is 71 (43 + 4 + 24) state times. This does not include the 12 state times required for PUSHF if it is used as the first instruction in the interrupt routine or additional latency caused by having the interrupt masked or disabled.

Interrupt latency time can be reduced by careful selection of instructions in areas of code where interrupts are expected. Using 'EI' followed immediately by a long instruction (e.g. MUL, NORML, etc.) will increase the maximum latency by 4 state times, as an interrupt cannot occur between EI and the instruction following EI. The 'DI', 'PUSHF', 'POPF' and 'TRAP' instructions will also cause the same situation. Typically the PUSHF, POPF and TRAP instructions would only effect latency when one interrupt routine is already in process, as these instructions are seldom used at other times.

## 2.6. TIMERS

Two 16-bit timers are available for use on the 8096. The first is designated 'Timer 1', the second, 'Timer 2'. Timer 1 is used to synchronize events to real time, while Timer 2 can be clocked externally and synchronizes events to external occurrences.

### 2.6.1. Timer 1

Timer 1 is clocked once every eight state times and can be cleared only by executing a reset. The only other way to change its value is by writing to 000CH but this is a test mode which sets both timers to 0FFFH and should not be used in programs.

### 2.6.2. Timer 2

Timer 2 can be incremented by transitions (one count each transition, rising and falling) on either T2CLK or HSI.1. The multiple functionality of the timer is determined by the state of I/O Control Register 0, bit 7 (IOC0.7). To ensure that all CAM entries are checked each count of Timer 2, the maximum transition speed is limited to once per eight state times. Timer 2 can be cleared by: executing a reset, by setting IOC0.1, by triggering HSO channel 0EH, or by pulling T2RST or HSI.0 high. The HSO and

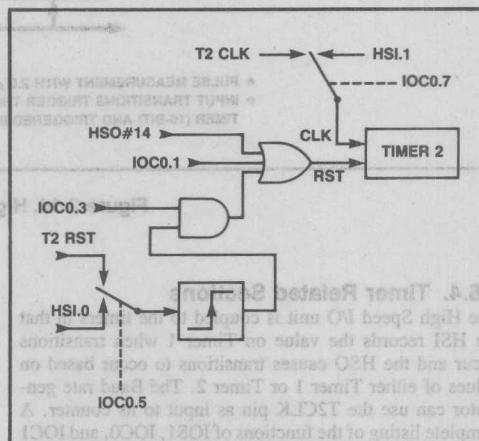


Figure 2-13. Timer 2 Clock and Reset Options

CAM are described in section 2.8. IOC0.3 and IOC0.5 control the resetting of Timer 2. Figure 2-13 shows the different ways of manipulating Timer 2.

### 2.6.3. Timer Interrupts

Both Timer 1 and Timer 2 can be used to trigger a timer overflow interrupt and set a flag in the I/O Status Register 1 (IOS1). The interrupts are controlled by IOC1.2 and IOC1.3 respectively. The flags are set in IOS1.5 and IOS1.4, respectively.

Caution must be used when examining the flags, as any access (including Compare and Jump on Bit) of IOS1 clears the whole byte, including the software timer flags. It is, therefore, recommended to write the byte to a temporary register before testing bits. The general enabling and disabling of the timer interrupts are controlled by the Interrupt Mask Register bit 0. In all cases, setting a bit enables a function, while clearing a bit disables it.

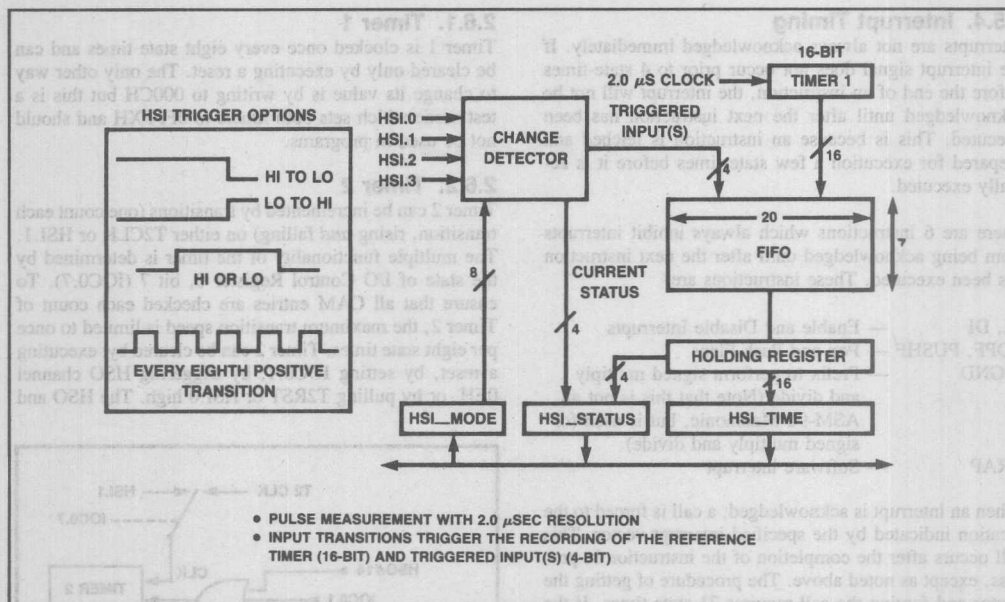


Figure 2-14. High Speed Input Unit

## 2.6.4. Timer Related Sections

The High Speed I/O unit is coupled to the timers in that the HSI records the value on Timer 1 when transitions occur and the HSO causes transitions to occur based on values of either Timer 1 or Timer 2. The Baud rate generator can use the T2CLK pin as input to its counter. A complete listing of the functions of IOS1, IOC0, and IOC1 are in section 2.13.

## 2.7. HIGH SPEED INPUTS

The High Speed Input Unit (HSI), can be used to record the time at which an event occurs with respect to Timer 1. There are 4 lines (HSI.0 through HSI.3) which can be used in this mode and up to a total of 8 events can be recorded. HSI.2 and HSI.3 share pins with HSO.4 and HSO.5. The I/O Control Registers (IOC0 and IOC1) are used to determine the functions of these pins. A block diagram of the HSI unit is shown in Figure 2-14.

### 2.7.1. HSI Modes

There are 4 possible modes of operation for each of the HSI. The HSI mode register is used to control which pins will look for what type of events. The 8-bit register is set up as shown in Figure 2-15.

High and low levels each need to be held for at least 1 state time to ensure proper operation. The maximum input speed is 1 event every 8 state times except when the 8 transition mode is used, in which case it is 1 transition per state time.

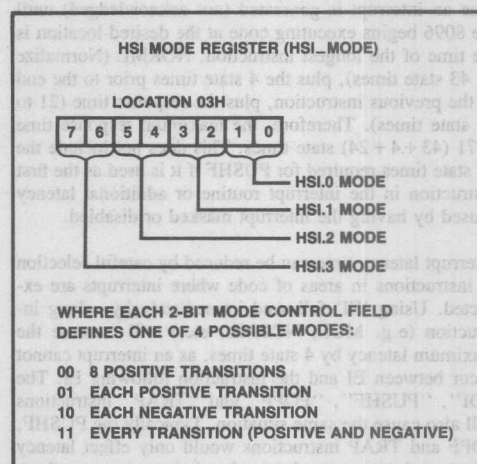
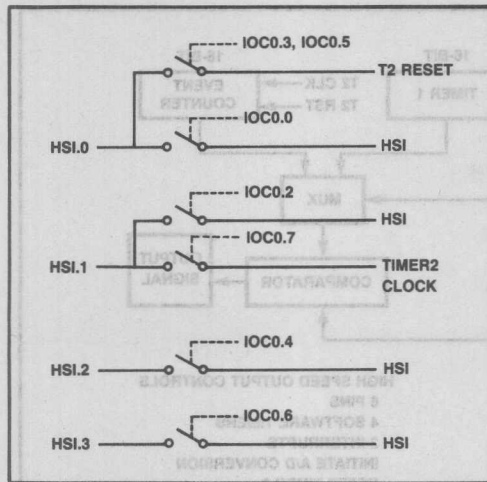


Figure 2-15. HSI Mode Register Diagram

The HSI lines can be individually enabled and disabled using bits in IOC0, at location 0015H. Figure 2-16 shows the bit locations which control the HSI pins. If the pin is disabled, transitions will not be entered in the FIFO.



**Figure 2-16. IOC0 Control of HSI Pin Functions**

## 2.7.2. HSI FIFO

When an HSI event occurs, a  $7 \times 20$  FIFO stores the 16 bits of Timer 1 and the 4 bits indicating the state of the 4 HSI lines at the time the status is read. It can take up to 8 state times for this information to reach the holding register. For this reason, 8 state times must be allowed between consecutive reads of HSI\_TIME. When the FIFO is full, one additional event for a total of 8 events can be stored by considering the holding register part of the FIFO. If the FIFO and holding register are full any additional events will not be recorded.

## 2.7.3. HSI Interrupts

Interrupts can be generated from the HSI unit in one of two ways, determined by IOC1.7. If the bit is a 0, then an interrupt will be generated every time a value is loaded into the holding register. If it is a 1, an interrupt will only be generated when the FIFO, (independent of the holding register), has six entries in it. Since all interrupts are rising edge triggered, if IOC1.7=1, the processor will not be re-interrupted until the FIFO first contains 5 or less records, then contains six or more. Interrupts can also be generated by pin HSI.0, which has its own interrupt vector.

## 2.7.4. HSI Status

Bits 6 and 7 of the I/O Status register 1 (IOS1) indicate the status of the HSI FIFO. If bit 6 is a 1, the FIFO contains at least six entries. If bit 7 is a 1, the FIFO contains at least 1 entry and the holding register has been loaded. The FIFO may be read after verifying that it contains valid data. Caution must be used when reading or testing bits in IOS1, as this action clears the entire byte, including the software and hardware timer overflow flags. It is best to store the byte and then test the stored value. See Section 3.7.2.

Reading the HSI is done in two steps. First, the HSI Status

register is read to obtain the current state of the HSI pins and which pins had changed at the recorded time. The format of the HSI\_STATUS Register is shown in Figure 2-17. Second, the HSI Time register is read. Reading the Time register unloads one word of the FIFO, so if the Time register is read before the Status register, the information in the Status register will be lost. The HSI Status register is at location 06H and the HSI Time registers are in locations 04H and 05H.

If the HSI\_TIME and Status register are read without the holding register being loaded, the values read will be undeterminate.

It should be noted that many of the Status register conditions are changed by a reset, see section 2.15.2. A complete listing of the functions of IOS0, IOS1, and IOC1 can be found in section 2.13.

## 2.8. HIGH SPEED OUTPUTS

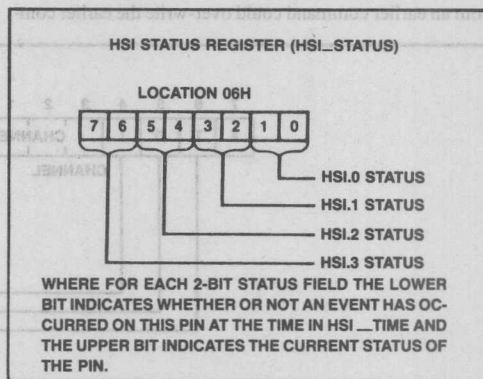
The High Speed Output unit (HSO) is used to trigger events at specific times with minimal CPU overhead. These events include: starting an A to D conversion, re-setting Timer 2, setting 4 software flags, and switching up to 6 output lines. Interrupts can be generated whenever one of these events is triggered. Up to 8 events can be pending at any one time.

### 2.8.1. HSO Shared Pins

Two of the 6 output lines (HSO.0 through HSO.5) are shared with the High Speed Input (HSI) lines. HSO.4 and HSO.5 are shared with HSI.2 and HSI.3, respectively. Bits 4 and 6 of the I/O Control Register 1 (IOC1) are used to enable HSO.4 and HSO.5 as outputs.

### 2.8.2. HSO CAM

A block diagram of the HSO unit is shown in Figure 2-18. The Content Addressable Memory (CAM) file is the center of control. One CAM register is compared with a time value every state time. Therefore, it takes 8 state times to compare all CAM registers with a timer.



**Figure 2-17. HSI Status Register Diagram**

## ARCHITECTURAL OVERVIEW

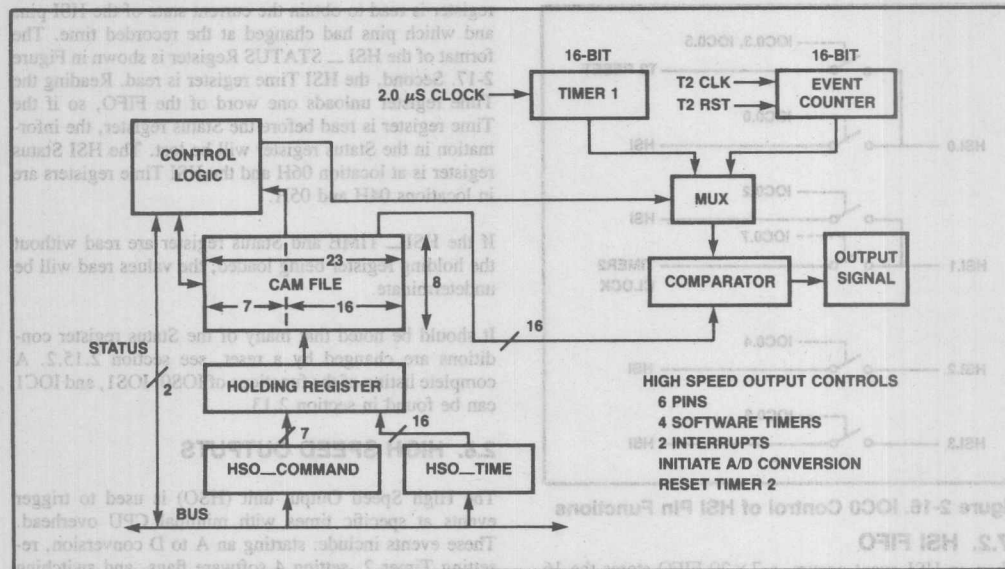


Figure 2-18. High Speed Output Unit

Each CAM register is 23 bits wide. Sixteen bits specify the time at which the action is to be carried out and 7 bits specify both the nature of the action and whether Timer 1 or Timer 2 is the reference. The format of the command to the HSO unit is shown in Figure 2-19. Note that bit 5 is ignored for command channels 8 through 0FH.

To enter a command into the CAM file, write the 7-bit "Command Tag" into location 0006H followed by the time at which the action is to be carried out into word address 0004H. Writing the time value loads the HSO Holding Register with both the time and the last written command tag. The command does not actually enter the CAM file until an empty CAM register becomes available. Since it can take up to 8 state times for a command to enter the CAM, commands written less than 8 state times from an earlier command could over-write the earlier com-

mand. In addition, if Timer 1 is being used as the reference, the minimum time that can be loaded is Timer 1 + 2. A similar restriction applies if Timer 2 is used as the reference.

Care must be taken when writing the command tag for the HSO. If an interrupt occurs during the time between writing the command tag and loading the time value, and the interrupt service routine writes to the HSO time register, the command tag used in the interrupt routine will be written to the CAM at both the time specified by the interrupt routine and the time specified by the main program. The command tag from the main program will not be executed. One way of avoiding this problem would be to disable interrupts when writing commands and times to the HSO unit. See also Section 3.7.3.

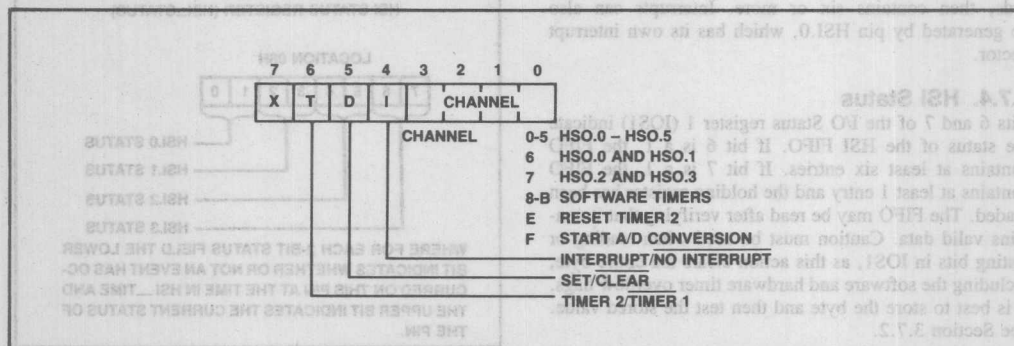


Figure 2-19. HSO Command Tag Format



### 2.8.3. HSO Status

Before writing to the HSO, it is desirable to ensure that the Holding Register is empty. If it is not, writing to the HSO will overwrite the value in the Holding Register. I/O Status Register 0 (IOS0) bits 6 and 7 indicate the status of the HSO unit. This register is described in section 2.13.4. If IOS0.6 equals 0, the holding register is empty and at least one CAM register is empty. If IOS0.7 equals 0, the holding register is empty.

One location in the CAM file is checked each state-time. Thus, it takes 8 state-times for the Holding Register to have had access to all 8 CAM registers. Similarly, it takes 8 state-times for the comparator to have had access to all 8 CAM registers. This defines the time-resolution of the HSO unit to be 8 state-times (2.0  $\mu$ sec, if the oscillator frequency is 12 MHz). Note that the comparator does not look at the holding register, so instructions in the holding register do not execute.

### 2.8.4. Clearing The HSO

All 8 CAM locations of the HSO are compared before any action is taken. This allows a pending external event to be cancelled by simply writing the opposite event to the CAM. However, once an entry is placed in the CAM, it cannot be removed until either the specified timer matches the written value or the chip is reset. Internal events are not synchronized to Timer 1, and therefore cannot be cleared. This includes events on HSO channels 8 through F and all interrupts. Since interrupts are not synchronized it is possible to have multiple interrupts at the same time value.

### 2.8.5. Using Timer 2 With The HSO

Timer 1 is incremented only once every 8 state-times. When it is being used as the reference timer for an HSO action, the comparator has a chance to look at all 8 CAM registers before Timer 1 changes its value. Following the same reasoning, Timer 2 has been synchronized to allow it to change at a maximum rate of once per 8 state-times. Timer 2 increments on both edges of the input signal.

When using Timer 2 as the HSO reference, caution must be taken that Timer 2 is not reset prior to the highest value for a Timer 2 match in the CAM. This is because the HSO CAM will hold an event pending until a time match occurs, if that match is to a time value on Timer 2 which is never reached, the event will remain pending in the CAM until the part is reset.

Additional caution must be used when Timer 2 is being reset using the HSO unit, since resetting Timer 2 using the HSO is an internal event and can therefore happen at any time within the eight-state-time window. For this reason, any events scheduled to occur at the same time as a Timer 2 reset should be logged into the CAM with a Timer 2 value of zero. When using this method to make a programmable modulo counter, the count will stay at the maximum Timer 2 value only until the Reset T2 command is recognized. The count will stay at zero for the transition which would have changed the count from "N" to zero, and then change to a one on the next transition.

### 2.8.6. Software Timers

The HSO can be programmed to generate interrupts at preset times. Up to four such "Software Timers" can be in operation at a time. As each preprogrammed time is reached, the HSO unit sets a Software Timer Flag. If the interrupt bit in the command tag was set then a Software Timer Interrupt will also be generated. The interrupt service routine can then examine I/O Status register 1 (IOS1) to determine which software timer expired and caused the interrupt. When the HSO resets Timer 2 or starts an A to D conversion, it can also be programmed to generate a software timer interrupt but there is no flag to indicate that this has occurred. See also Section 3.7.4.

If more than one software timer interrupt occurs in the same time frame it is possible that multiple software timer interrupts will be generated.

Each read or test of any bit in IOS1 will clear the whole byte. Be certain to save the byte before testing it unless you are only concerned with 1 bit. See also Section 3.2.2.

A complete listing of the functions of IOS0, IOS1, and IOC1 can be found in section 2.13. The Timers are described in section 2.6 and the HSI is described in section 2.7.

## 2.9. ANALOG INPUTS

The A to D converter on the 8096 provides a 10-bit result on one of 8 input channels. Conversion is done using successive approximation with a result equal to the ratio of the input voltage divided by the analog supply voltage. If the ratio is 1.00, then the result will be all ones. The A/D converter is available on the 8097, 8397, 8095 and 8395 members of the MCS<sup>®</sup>-96 family. The A/D converter on the 8096BH is slightly different than that on the 8096, see the data sheet in Chapter 5.

### 2.9.1. A/D Accuracy

Each conversion requires 168 state-times (42  $\mu$ s at 12 MHz) independent of the accuracy desired or value of input voltage. The input voltage must be in the range of 0 to VREF, the analog reference and supply voltage. For proper operation, VREF (the reference voltage and analog power supply) must be held at  $VCC \pm 0.3V$  with  $VREF = 5.0 \pm 0.5V$ . The A/D result is calculated from the formula:

$$1023 \times (\text{input voltage} - \text{ANGND}) / (VREF - \text{ANGND})$$

It can be seen from this formula that changes in VREF or ANGND effect the output of the converter. This can be advantageous if a ratiometric sensor is used since these sensors have an output that can be measured as a proportion of VREF.

If high absolute accuracy is needed it may be desirable to use a separate power supply, or power traces, to operate the A/D converter. There is no sample and hold circuit internal to the chip, so the input voltage must be held constant for the entire 168 state times. Examples of connecting the A/D converter to various devices are given in section 4.3.

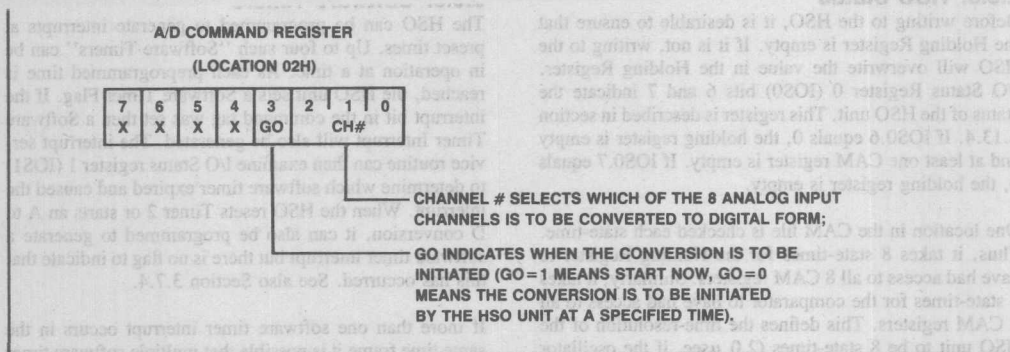


Figure 2-20. A/D Command Register

### 2.9.2. A/D Commands

Analog signals can be sampled by any one of the 8 analog input pins (ACH0 through ACH7) which are shared with Port 0. ACH7 can also be used as an external interrupt if IOC1.1 is set (see section 2.5). The A/D Command Register, at location 02H, selects which channel is to be converted and whether the conversion should start immediately or when the HSO (Channel #0FH) triggers it. The A/D command register must be written for each conversion, even if the HSO is used as the trigger. A to D commands are formatted as shown in Figure 2-20.

The command register is double buffered so it is possible to write a command to start a conversion triggered by the HSO while one is still in progress. Care must be taken when this is done since if a new conversion is started while one is already in progress, the conversion in progress is cancelled and the new one is started. When a conversion is started, the result register is cleared. For this reason the result register must be read before a new conversion is started or data will be lost.

### 2.9.3. A/D Results

Results of the analog conversions are read from the A/D

Result Register at locations 02H and 03H. Although these addresses are on a word boundary, they must be read as individual bytes. Information in the A/D Result register is formatted as shown in Figure 2-21. Note that the status bit may not be set until 8 state times after the go command. Information on using the HSO is in section 2.8.

### 2.10. PULSE WIDTH MODULATION OUTPUT (D/A)

Digital to analog conversion can be done with the pulse width modulation output; a block diagram of the circuit is shown in Figure 2-22. The 8-bit counter is incremented every state time. When it equals 0, the PWM output is set to a one. When the counter matches the value in the PWM register, the output is switched low. When the counter overflows, the output is once again switched high. A typical output waveform is shown in Figure 2-23. Note that when the PWM register equals 00, the output is always low.

The output waveform is a variable duty cycle pulse which repeats every 256 state times (64  $\mu$ S at 12MHz). Changes

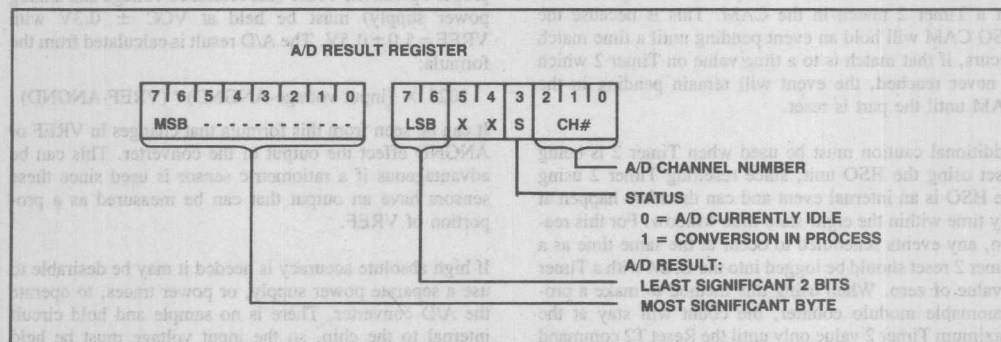


Figure 2-21. A/D Result Register

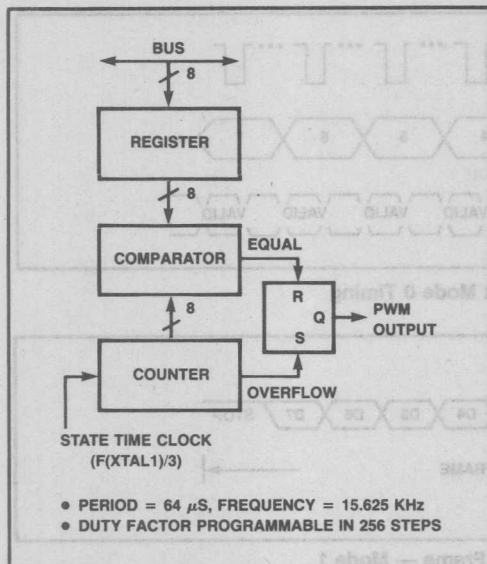


Figure 2-22. Pulse Width Modulated (D/A) Output

in the duty cycle are made by writing to the PWM register at location 17H. There are several types of motors which require a PWM waveform for most efficient operation. Additionally, if this waveform is integrated it will produce a DC level which can be changed in 256 steps by varying the duty cycle.

Details about the hardware required for smooth, accurate D/A conversion can be found in section 4.3.2. Typically,

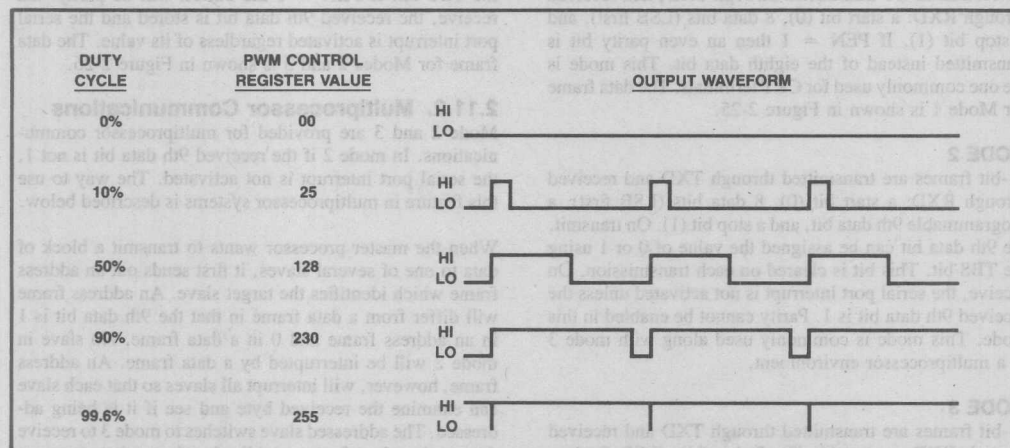


Figure 2-23. Typical PWM Outputs

some form of buffer and integrator are needed to obtain the most usefulness from this feature.

The PWM output shares a pin with Port 2, pin 5 so that these two features cannot be used at the same time. IOC1.0 equal to 1 selects the PWM function instead of the standard port function. More information on IOC1 is in section 2.13.3.

## 2.11. SERIAL PORT

The serial port is compatible with the MCS-51 serial port. It is full duplex, meaning it can transmit and receive simultaneously. It is also receive-buffered, meaning it can commence reception of a second byte before a previously received byte has been read from the receive register. The serial port registers (SBUF) are both accessed at location 07H. A write to this location accesses the transmit register, and a read accesses a physically separate receive register.

The serial port can operate in 4 modes (explained below). Selection of these modes is done through the Serial Port Status/Control register at location 11H, shown in Figure 2-27.

### 2.11.1. Serial Port Modes

#### MODE 0

Mode 0 is a shift register mode. The 8096 outputs a train of 8 shift pulses to an external shift register to clock 8 bits of data into or out of the register from or to the 8096. Serial data enters and exits the 8096 through RXD. TXD outputs the shift clock. 8 bits are transmitted or received, LSB first. A timing diagram of this mode is shown in Figure 2-24. This mode is useful as an I/O expander in which application external shift registers can be used as additional parallel I/O ports. An example of using the port in this mode is given in section 4.5.

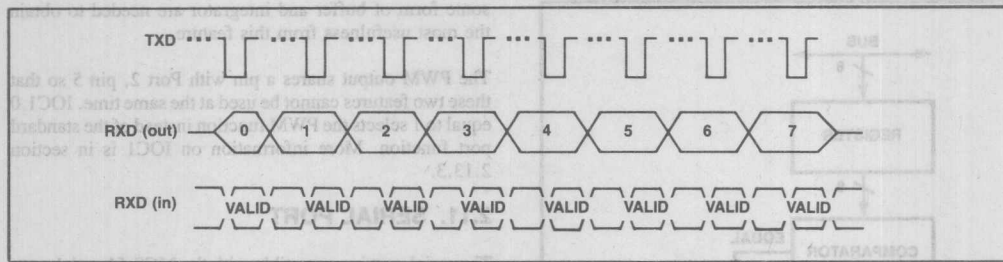


Figure 2-24. Serial Port Mode 0 Timing

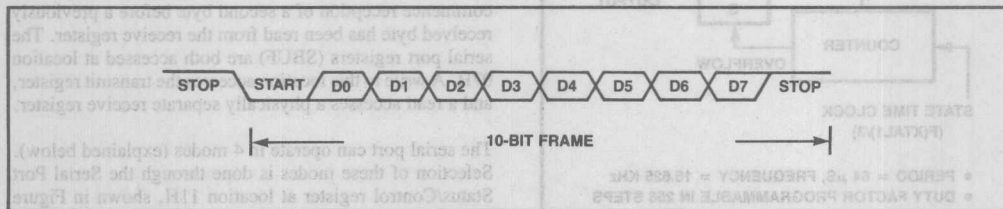


Figure 2-25. Serial Port Frame — Mode 1

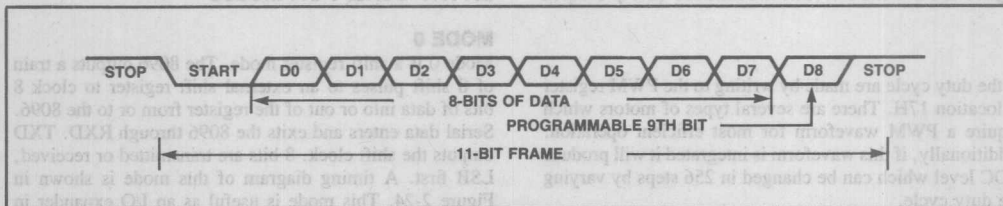


Figure 2-26. Serial Port Frame Modes 2 and 3

## MODE 1

10-bit frames are transmitted through TXD, and received through RXD: a start bit (0), 8 data bits (LSB first), and a stop bit (1). If PEN = 1 then an even parity bit is transmitted instead of the eighth data bit. This mode is the one commonly used for CRT terminals. The data frame for Mode 1 is shown in Figure 2-25.

## MODE 2

11-bit frames are transmitted through TXD and received through RXD: a start bit (0), 8 data bits (LSB first), a programmable 9th data bit, and a stop bit (1). On transmit, the 9th data bit can be assigned the value of 0 or 1 using the TB8 bit. This bit is cleared on each transmission. On receive, the serial port interrupt is not activated unless the received 9th data bit is 1. Parity cannot be enabled in this mode. This mode is commonly used along with mode 3 in a multiprocessor environment.

## MODE 3

11-bit frames are transmitted through TXD and received through RXD: a start bit (0), 8 data bits (LSB first), a programmable 9th data bit, and a stop bit (1). On transmit,

the 9th data bit can be assigned the value of 0 or 1 using the TB8 bit. If PEN = 1 the 9th bit will be parity. On receive, the received 9th data bit is stored and the serial port interrupt is activated regardless of its value. The data frame for Modes 2 and 3 is shown in Figure 2-26.

## 2.11.2. Multiprocessor Communications

Mode 2 and 3 are provided for multiprocessor communications. In mode 2 if the received 9th data bit is not 1, the serial port interrupt is not activated. The way to use this feature in multiprocessor systems is described below.

When the master processor wants to transmit a block of data to one of several slaves, it first sends out an address frame which identifies the target slave. An address frame will differ from a data frame in that the 9th data bit is 1 in an address frame and 0 in a data frame. No slave in mode 2 will be interrupted by a data frame. An address frame, however, will interrupt all slaves so that each slave can examine the received byte and see if it is being addressed. The addressed slave switches to mode 3 to receive the coming data frames, while the slaves that were not addressed stay in mode 2 and go on about their business.



### 2.11.3. Controlling the Serial Port

Control of the Serial Port is done through the Serial Port Control/Status register. The format for the control word is shown in Figure 2-27. Note that reads access only part of the byte, as do writes, and that TB8 is cleared after each byte is transmitted.

In Mode 0, if REN=0, writing to SBUF will start a transmission. Causing a rising edge on REN, or clearing RI with REN = 1, will start a reception. Setting REN=0 will stop a reception in progress, and inhibit further receptions. To avoid a partial or complete undesired reception, REN must be set to zero before clearing RI. This can be handled in an interrupt environment by using software flags, or in a straight-line code environment by using the Interrupt Pending register to signal the completion of a receive. In any mode, it is necessary to set IOC1.5 to a 1 to enable the TXD pin. Some examples of the software involved in using the serial port can be found in section 3.8. More information on IOC1 is in section 2.13.3.

#### 2.11.4. Determining Baud Rates

Baud rates in all modes are determined by the contents of a 16-bit register at location 000EH. This register must be loaded sequentially with 2 bytes (least significant byte first). The MSB of this register selects one of two sources for the input frequency to the baud rate generator. If it is a 1, the frequency on the XTAL1 pin is selected, if not, the external frequency from the T2CLK pin is used. It should be noted that the maximum speed of T2CLK is one transition every 2 state times, with a minimum period of 16 XTAL1 cycles.

The unsigned integer represented by the lower 15 bits of the baud rate register defines a number B, where B has a maximum value of 32767. The baud rate for the four serial modes using either XTAL1 or T2CLK as the clock source is given by:

Using XTAL1:

$$\text{Mode 0: } \frac{\text{Baud Rate}}{4 * (B + 1)} = \frac{\text{XTAL1 frequency}}{4 * (B + 1)}, B \neq 0$$

Others:  $\text{Baud Rate} = \frac{\text{XTAL1 frequency}}{64*(B + 1)}$

Using T2CLK:

$$\text{Mode 0: } \frac{\text{Baud Rate}}{\text{Rate}} = \frac{\text{T2CLK frequency}}{B}; B \neq 0$$

Others:  $\text{Baud Rate} = \frac{\text{T2CLK frequency}}{16 * B}$ ;  $B \neq 0$

Note that B cannot equal 0, except when using XTAL1 in other than mode 0.

Common baud rate values, using XTAL1 at 12MHz, are shown below.

| Baud Rate | Baud Register<br>Mode 0 | Value<br>Others |
|-----------|-------------------------|-----------------|
| 9600      | 8138H                   | 8014H           |
| 4800      | 8271H                   | 8027H           |
| 2400      | 84E2H                   | 804EH           |
| 1200      | 89C4H                   | 809CH           |
| 300       | A710H                   | 8271H           |

The maximum asynchronous baud rate is 187.5 Kbaud, with a 12MHz clock.

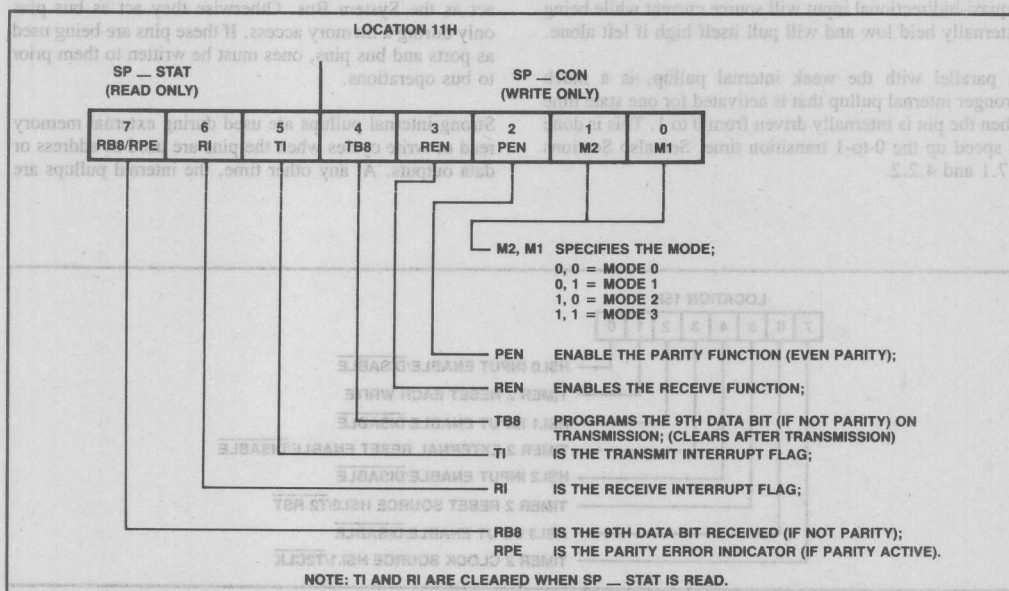


Figure 2-27. Serial Port Control/Status Register

There are five 8-bit I/O ports on the 8096. Some of these ports are input only, some output only, some bidirectional and some have alternate functions. Input ports connect to the internal bus through an input buffer. Output ports connect through an output buffer to an internal register that holds the output bits. Bidirectional ports consist of an internal register, an output buffer, and an input buffer.

When an instruction accesses a bidirectional port as a source register, the question often arises as to whether the value that is brought into the CPU comes from the internal port register or from the port pins through the input buffer. In the 8096, the value always comes from the port pins, never from the internal register.

### 2.12.1. Port 0

Port 0 is an input only port which shares its pins with the analog inputs to the A/D Converter. One can read Port 0 digitally, and/or, by writing the appropriate control bits to the A/D Command Register, select one of the lines of this port to be the input to the A/D Converter.

### 2.12.2. Port 1

Port 1 is a quasi-bidirectional I/O port. "Quasi-bidirectional" means the port pin has a weak internal pullup that is always active and an internal pulldown which can either be on (to output a 0) or off (to output a 1). If the internal pulldown is left off (by writing a 1 to the pin), the pin's logic level can be controlled by an external pulldown which can either be on (to input a 0) or off (to input a 1). From the user's point of view the main distinction is that a quasi-bidirectional input will source current while being externally held low and will pull itself high if left alone.

In parallel with the weak internal pullup, is a much stronger internal pullup that is activated for one state time when the pin is internally driven from 0 to 1. This is done to speed up the 0-to-1 transition time. See also Sections 3.7.1 and 4.2.2.

Port 2 is a multi-functional port. Six of its pins are shared with other functions in the 8096, as shown below.

| Port | Function            | Alternate Function           | Controlled by |
|------|---------------------|------------------------------|---------------|
| P2.0 | output              | TXD (serial port transmit)   | IOC1.5        |
| P2.1 | input               | RXD (serial port receive)    | N/A           |
| P2.2 | input               | EXTINT (external interrupt)  | IOC1.1        |
| P2.3 | input               | T2CLK (Timer 2 input)        | IOC0.7        |
| P2.4 | input               | T2RST (Timer 2 reset)        | IOC0.5        |
| P2.5 | output              | PWM (pulse-width modulation) | IOC1.0        |
| P2.6 | quasi-bidirectional |                              |               |
| P2.7 | quasi-bidirectional |                              |               |

### 2.12.4. Ports 3 and 4 / AD 0-15

These pins have two functions. They are either bidirectional ports with open-drain outputs or System Bus pins which the memory controller uses when it is accessing off-chip memory. If the EA line is low, the pins always act as the System Bus. Otherwise they act as bus pins only during a memory access. If these pins are being used as ports and bus pins, ones must be written to them prior to bus operations.

Strong internal pullups are used during external memory read or write cycles when the pins are used as address or data outputs. At any other time, the internal pullups are

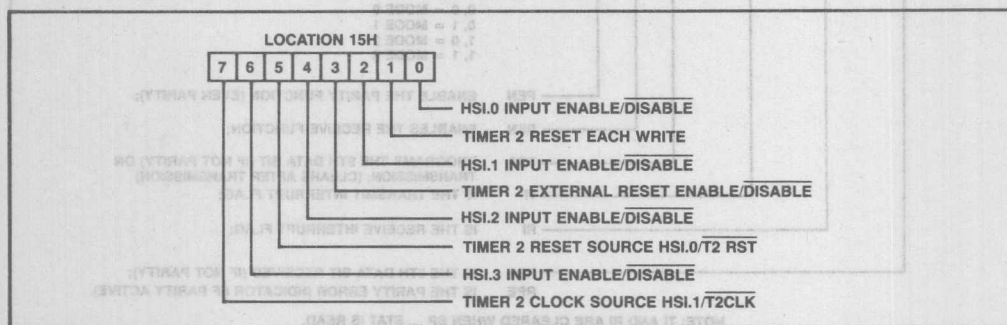


Figure 2-28. I/O Control Register 0 (IOC0)

disabled. The port pins and their system bus functions are shown below:

| Port Pin | System Bus Function |
|----------|---------------------|
| P3.0     | AD0                 |
| P3.1     | AD1                 |
| P3.2     | AD2                 |
| P3.3     | AD3                 |
| P3.4     | AD4                 |
| P3.5     | AD5                 |
| P3.6     | AD6                 |
| P3.7     | AD7                 |
| P4.0     | AD8                 |
| P4.1     | AD9                 |
| P4.2     | AD10                |
| P4.3     | AD11                |
| P4.4     | AD12                |
| P4.5     | AD13                |
| P4.6     | AD14                |
| P4.7     | AD15                |

## 2.13. STATUS AND CONTROL REGISTERS

### 2.13.1. I/O Control Registers

There are two I/O Control registers, IOC0 and IOC1. IOC0 controls Timer 2 and the HSI lines. IOC1 controls some pin functions, interrupt sources and 2 HSO pins.

Whenever input lines are switched between two sources, or enabled, it is possible to generate transitions on these lines. This could cause problems with respect to edge sensitive lines such as the HSI lines, Interrupt line, and Timer 2 control lines.

### 2.13.2. I/O Control Register 0 (IOC0)

IOC0 is located at 0015H. The four HSI lines can be enabled or disabled to the HSI unit by setting or clearing bits in IOC0. Timer 2 functions including clock and reset sources are also determined by IOC0. The control bit locations are shown in Figure 2-28.

### 2.13.3. I/O Control Register 1 (IOC1)

IOC1 is used to select some pin functions and enable or disable some interrupt sources. Its location is 0016H. Port pin P2.5 can be selected to be the PWM output instead of a standard output. The external interrupt source can be selected to be either EXTINT (same pin as P2.2) or Analog Channel 7 (ACH7, same pin as P0.7). Timer 1 and Timer 2 overflow interrupts can be individually enabled or disabled. The HSI interrupt can be selected to activate either when there is 1 FIFO entry or 7. Port pin P2.0 can be selected to be the TXD output. HSO.4 and HSO.5 can be enabled or disabled to the HSO unit. More information on interrupts is available in section 2.5. The positions of the IOC1 control bits are shown in Figure 2-29.

### 2.13.4. I/O Status Register 0 (IOS0)

There are two I/O Status registers, IOS0 and IOS1. IOS0, located at 0015H, holds the current status of the HSO lines and CAM. The status bits of IOS0 are shown in Figure 2-30.

### 2.13.5. I/O Status Register 1 (IOS1)

IOS1 is located at 0016H. It contains status bits for the timers and the HSI. Every access of this register clears all of the timer overflow and timer expired bits. It is, therefore, important to first store the byte in a temporary location before attempting to test any bit unless only one bit will ever be of importance to the program. The status bits of IOS1 are shown in Figure 2-31.

## 2.14. WATCHDOG TIMER (WDT)

This feature is provided as a means of graceful recovery from a software upset. Once the watchdog is initialized, if the software fails to reset the watchdog at least every 64K state times, a hardware reset will be initiated.

The watchdog is initialized by the first write of the clear WDT code to the WDT register. Once the watchdog is initialized it cannot be disabled by software. On reset the watchdog is not active.

The 8096BH has improved resistance to noise on the watchdog enable bit. See Chapter 5.

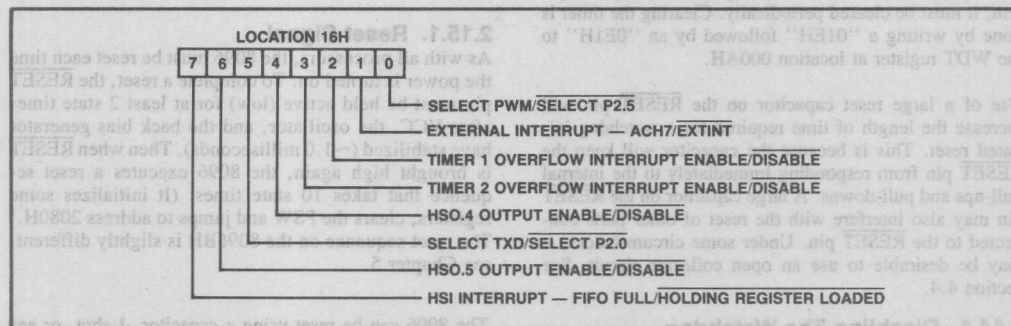


Figure 2-29. I/O Control Register 1 (IOC1)

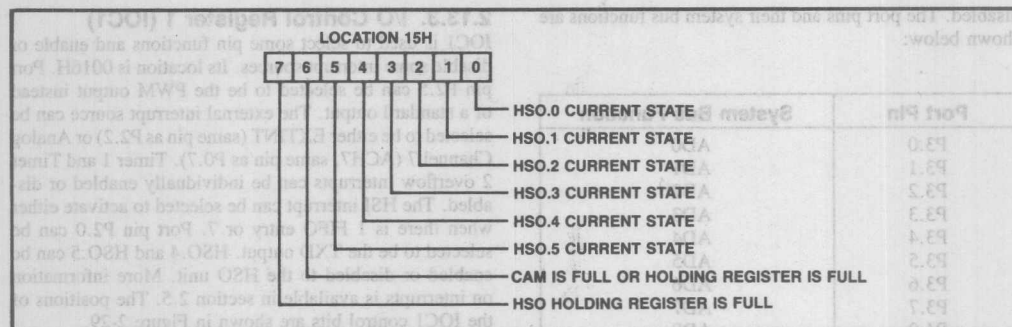


Figure 2-30. I/O Status Register 0 (IOS0)

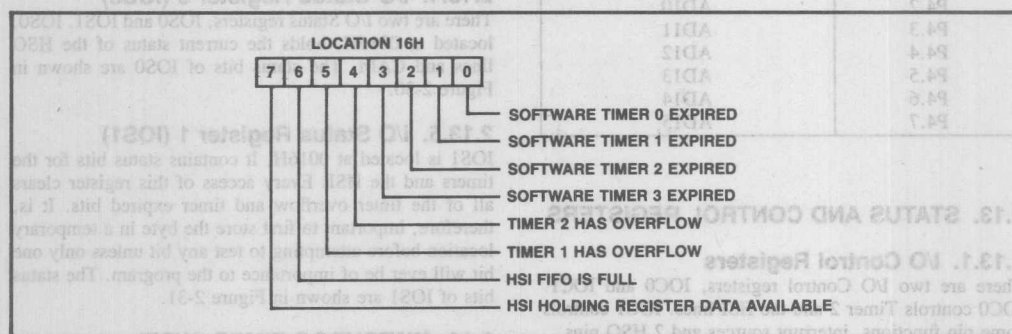


Figure 2-31. HSIO Status Register 1 (IOS1)

The software can be designed so that the watchdog times out if the program does not progress properly. The watchdog will also time-out if the software error was due to ESD (Electrostatic Discharge) or other hardware related problems. This prevents the controller from having a malfunction for longer than 16 mS if a 12 MHz oscillator is used.

The watchdog timer is a 16-bit counter which is incremented every state time. When it overflows it pulls down the RESET pin for at least two state times, resetting the 8096 and any other chips connected to the reset line. To prevent the timer from overflowing and resetting the system, it must be cleared periodically. Clearing the timer is done by writing a "01EH" followed by an "0E1H" to the WDT register at location 000AH.

Use of a large reset capacitor on the RESET pin will increase the length of time required for a watchdog initiated reset. This is because the capacitor will keep the RESET pin from responding immediately to the internal pull-ups and pull-downs. A large capacitor on the RESET pin may also interfere with the reset of other parts connected to the RESET pin. Under some circumstances, it may be desirable to use an open collector circuit. See section 4.4.

## 2.14.1. Disabling The Watchdog

The watchdog should be disabled by software not initial-

izing it. If this is not possible, such as during program development, the watchdog can be disabled by holding the RESET pin at 2.0 to 2.5 volts. Voltages over 2.5 volts on the pin could quickly damage the part. Even at 2.5 volts, using this technique for other than debugging purposes is not recommended, as it may effect long term reliability. It is further recommended that any part used in this way for more than several seconds, not be used in production versions of products.

## 2.15. RESET

### 2.15.1. Reset Signal

As with all processors, the 8096 must be reset each time the power is turned on. To complete a reset, the RESET pin must be held active (low) for at least 2 state times after VCC, the oscillator, and the back bias generator have stabilized (~1.0 milliseconds). Then when RESET is brought high again, the 8096 executes a reset sequence that takes 10 state times. (It initializes some registers, clears the PSW and jumps to address 2080H.) The reset sequence on the 8096BH is slightly different, see Chapter 5.

The 8096 can be reset using a capacitor, 1-shot, or any other method capable of providing a pulse of at least 2



state times longer than required for VCC and the oscillator to stabilize.

For best functionality, it is suggested that the reset pin be pulled low with an open collector device. In this way, several reset sources can be wire or'd together. Remember, the RESET pin itself can be a reset source (see section 2.14). Details of hardware suggestions for reset can be found in section 4.4.

## 2.15.2. Reset Status

The I/O lines and control lines of the 8096 will be in their reset state within 2 state times after reset is low, with VCC and the oscillator stabilized. Prior to that time, the status of the I/O lines is indeterminate. After the 10 state time reset sequence, the Special Function Registers will be set as follows:

| Register               | reset value |
|------------------------|-------------|
| Port 1                 | 11111111B   |
| Port 2                 | 110XXXX1B   |
| Port 3                 | 11111111B   |
| Port 4                 | 11111111B   |
| PWM Control            | 00H         |
| Serial Port (Transmit) | undefined   |
| Serial Port (Receive)  | undefined   |
| Baud Rate Register     | undefined   |
| Serial Port Control    | XXXX0XXXB   |
| Serial Port Status     | X00XXXXXB   |
| A/D Command            | undefined   |
| A/D Result             | undefined   |
| Interrupt Pending      | undefined   |
| Interrupt Mask         | 00000000B   |
| Timer 1                | 0000H       |
| Timer 2                | 0000H       |
| Watchdog Timer         | 0000H       |
| HSI Mode               | 11111111B   |
| HSI Status             | undefined   |
| IOS0                   | 00000000B   |
| IOS1                   | 00000000B   |
| IOC0                   | X0X0X0X0B   |
| IOC1                   | X0X0XXX1B   |
| HSI FIFO               | empty       |
| HSO CAM                | empty       |
| HSO lines              | 000000B     |
| PSW                    | 0000H       |
| Stack Pointer          | undefined   |
| Program Counter        | 2080H       |

Other conditions following a reset are:

| Pin  | reset value |
|------|-------------|
| RD   | high        |
| WR   | high        |
| ALE  | low         |
| BHE  | low         |
| INST | high        |

It is important to note that the Stack Pointer and Interrupt Pending Register are undefined, and need to be initialized in software. The Interrupts are disabled by both the mask register and PSW.9 after a reset.

## 2.15.3. Reset Sync Mode

The RESET line can be used to start the 8096 at an exact state time to provide for synchronization of test equipment and multiple chip systems. RESET is active low. To synchronize parts, RESET is brought high on the rising edge of XTAL1. Complete details on synchronizing parts can be found in section 4.1.5.

It is very possible that parts which start in sync may not stay that way. The best example of this would be when a "jump on I/O bit" is being used to hold the processor in a loop. If the line changes during the time it is being tested, one processor may see it as a one, while the other sees it as a zero. The result is that one processor will do an extra loop, thus putting it several states out of sync with the other.

## 2.16. PIN DESCRIPTION

On the 48-pin part the following pins are not bonded out: Port1, Port0 (Analog In) bits 0-3, T2CLK (P2.3), T2RST (P2.4), P2.6, P2.7, CLKOUT, INST, NMI, TEST (BUS-WIDTH on 8096BH).

See the 8096BH data sheet for pin descriptions of that series of parts.

### VCC

Main supply voltage (5V).

### VSS

Digital circuit ground (0V). There are two VSS pins, both must be tied to ground.

### VPD

RAM standby supply voltage (5V). This voltage must be present during normal operation. See section 2.4.2 and 4.

### VREF

Reference voltage and power supply for the analog portion of the A/D converter. Nominally at 5 volts. See section 2.9.1 and 4.

### ANGND

Reference ground for the A/D converter. Should be held at nominally the same potential as VSS. See section 2.9.1 and 4.

### VBB

Substrate voltage from the on-chip back-bias generator. This pin should be connected to ANGND through a 0.01  $\mu$ f capacitor (and not connected to anything else). The capacitor is not required if the A/D converter is not being used and the pin should be left floating.

### XTAL1

Input of the oscillator inverter and input to the internal clock generator. See sections 2.2 and 4.

## A1ALZ

Output of the oscillator inverter. See section 2.2.

## CLKOUT

Output of the internal clock generator. The frequency of CLKOUT is  $\frac{1}{2}$  the oscillator frequency. It has a 33% duty cycle. CLKOUT can drive one TTL input. See section 2.2.

## RESET

Reset input to the chip, also output to other circuits. Input low for at least 2 state times to reset the chip. RESET has a strong internal pullup. See section 2.15 and 4.1.

## TEST

Input low enables a factory test mode. The user should tie this pin to VCC for normal operation.

## NMI

A low to high transition causes a vector to external memory location 0000H. Reserved for use in Intel Development systems.

## INST

Output high while the address is valid during an external read indicates the read is an instruction fetch. See section 2.3.6 and 4.6.

## EA

Input for memory select (External Access).  $\overline{EA} = 1$  causes memory accesses to locations 2000H through 3FFFH to be directed to on-chip ROM.  $\overline{EA} = 0$  causes accesses to these locations to be directed to off-chip memory. EA has an internal pulldown, so it goes to 0 unless driven to 1. See section 2.3.3.

## ALE

Address Latch Enable output. ALE is activated only during external memory accesses. It is used to latch the address from the multiplexed address/data bus. See section 2.3.5 and 4.6.

## RD

Read signal output to external memory. RD is activated only during external memory reads. See section 2.3.5 and 4.6.

## WR

Write signal output to external memory. WR is activated only during external memory writes. See section 2.3.6 and 4.6.

## BHE

Bus High Enable signal output to external memory. BHE (0/1) selects/deselects the bank of memory that is connected to the high byte of the data bus. See section 2.3.5 and 4.6.

## READY

The READY input is used to lengthen external memory bus cycles up to the time specified in the data sheet. It has a weak internal pullup. See section 2.3.6 and 4.6.

## HSI

High impedance inputs to HSI Unit. Four HSI pins are available: HSI.0, HSI.1, HSI.2, and HSI.3. Two of them (HSI.2 and HSI.3) are shared with the HSO Unit. See section 2.7.

## HSO

Outputs from HSO Unit. Six HSO pins are available: HSO.0, HSO.1, HSO.2, HSO.3, HSO.4, and HSO.5. Two of them (HSO.4 and HSO.5) are shared with the HSI Unit. All HSO pins are capable of driving one TTL input. See section 2.8.

## PORT 0/ANALOG CHANNEL

High impedance input-only port. These pins can be used as digital inputs and/or as analog inputs to the on-chip A/D converter. See sections 2.9 and 2.12.1.

## PORT 1

Quasi-bidirectional I/O port. All pins of P1 are capable of driving one LS TTL input. See section 2.12.2.

## PORT 2

Multi-functional port. Six of its pins are shared with other functions in the 8096, as shown below.

| Port | Function            | Alternate Function          | Reference section |
|------|---------------------|-----------------------------|-------------------|
| P2.0 | output              | TXD (serial port transmit)  | 2.11.3            |
| P2.1 | input               | RXD (serial port receive)   | 2.11.3            |
| P2.2 | input               | EXTINT (external interrupt) | 2.5               |
| P2.3 | input               | T2CLK (Timer 2 input)       | 2.6.2             |
| P2.4 | input               | T2RST (Timer 2 reset)       | 2.6.2             |
| P2.5 | output              | PWM (pulse-width modulator) | 2.10              |
| P2.6 | quasi-bidirectional |                             |                   |
| P2.7 | quasi-bidirectional |                             |                   |

## ARCHITECTURAL OVERVIEW

The multi-functional inputs are high impedance. See section 2.12.3.

### PORTS 3 AND 4 / AD 0-15

These pins are shared between two 8-bit I/O ports and the multiplexed address/data bus. They are open drain except when being used as system bus pins. See section 2.3.5.

### 2.17. PIN LIST

The following is a list of pins in alphabetical order. Where a pin has two names it has been listed under both names, except for the system bus pins, AD0-AD15, which are listed under Port 3 and Port 4.

| Name        | 68-Pin<br>PLCC | 68-Pin<br>PGA | 48-Pin<br>DIP |
|-------------|----------------|---------------|---------------|
| ACH0/P0.0   | 6              | 4             | —             |
| ACH1/P0.1   | 5              | 5             | —             |
| ACH2/P0.2   | 7              | 3             | —             |
| ACH3/P0.3   | 4              | 6             | —             |
| ACH4/P0.4   | 11             | 67            | 43            |
| ACH5/P0.5   | 10             | 68            | 42            |
| ACH6/P0.6   | 8              | 2             | 40            |
| ACH7/P0.7   | 9              | 1             | 41            |
| ALE         | 62             | 16            | 34            |
| ANGND       | 12             | 66            | 44            |
| BHE         | 41             | 37            | 15            |
| CLKOUT      | 65             | 13            | —             |
| EA          | 2              | 8             | 39            |
| EXTINT/P2.2 | 15             | 63            | 47            |
| HSI.0       | 24             | 54            | 3             |
| HSI.1       | 25             | 53            | 4             |
| HSI.2/HSO.4 | 26             | 52            | 5             |
| HSI.3/HSO.5 | 27             | 51            | 6             |
| HSO.0       | 28             | 50            | 7             |
| HSO.1       | 29             | 49            | 8             |
| HSO.2       | 34             | 44            | 9             |
| HSO.3       | 35             | 43            | 10            |
| HSO.4/HSI.2 | 26             | 52            | 5             |
| HSO.5/HSI.3 | 27             | 51            | 6             |
| INST        | 63             | 15            | —             |
| NMI         | 3              | 7             | —             |
| PWM/P2.5    | 39             | 39            | 13            |
| P0.0/ACH0   | 6              | 4             | —             |
| P0.1/ACH1   | 5              | 5             | —             |
| P0.2/ACH2   | 7              | 3             | —             |
| P0.3/ACH3   | 4              | 6             | —             |
| P0.4/ACH4   | 11             | 67            | 43            |
| P0.5/ACH5   | 10             | 68            | 42            |
| P0.6/ACH6   | 8              | 2             | 40            |
| P0.7/ACH7   | 9              | 1             | 41            |
| P1.0        | 19             | 59            | —             |

| Name        | 68-Pin<br>PLCC | 68-Pin<br>PGA | 48-Pin<br>DIP |
|-------------|----------------|---------------|---------------|
| P1.1        | 20             | 58            | —             |
| P1.2        | 21             | 57            | —             |
| P1.3        | 22             | 56            | —             |
| P1.4        | 23             | 55            | —             |
| P1.5        | 30             | 48            | —             |
| P1.6        | 31             | 47            | —             |
| P1.7        | 32             | 46            | —             |
| P2.0/TXD    | 18             | 60            | 2             |
| P2.1/RXD    | 17             | 61            | 1             |
| P2.2/EXTINT | 15             | 63            | 47            |
| P2.3/T2CLK  | 44             | 34            | —             |
| P2.4/T2RST  | 42             | 36            | —             |
| P2.5/PWM    | 39             | 39            | 13            |
| P2.6        | 33             | 45            | —             |
| P2.7        | 38             | 40            | —             |
| P3.0/AD0    | 60             | 18            | 32            |
| P3.1/AD1    | 59             | 19            | 31            |
| P3.2/AD2    | 58             | 20            | 30            |
| P3.3/AD3    | 57             | 21            | 29            |
| P3.4/AD4    | 56             | 22            | 28            |
| P3.5/AD5    | 55             | 23            | 27            |
| P3.6/AD6    | 54             | 24            | 26            |
| P3.7/AD7    | 53             | 25            | 25            |
| P4.0/AD8    | 52             | 26            | 24            |
| P4.1/AD9    | 51             | 27            | 23            |
| P4.2/AD10   | 50             | 28            | 22            |
| P4.3/AD11   | 49             | 29            | 21            |
| P4.4/AD12   | 48             | 30            | 20            |
| P4.5/AD13   | 47             | 31            | 19            |
| P4.6/AD14   | 46             | 32            | 18            |
| P4.7/AD15   | 45             | 33            | 17            |
| RD          | 61             | 17            | 33            |
| READY       | 43             | 35            | 16            |
| RESET       | 16             | 62            | 48            |
| RXD/P2.1    | 17             | 61            | 1             |
| TEST        | 64             | 14            | —             |
| TXD/P2.0    | 18             | 60            | 2             |
| T2CLK/P2.3  | 44             | 34            | —             |
| T2RST/P2.4  | 42             | 36            | —             |
| VBB         | 37             | 41            | 12            |
| VCC         | 1              | 9             | 38            |
| VPD         | 14             | 64            | 46            |
| VREF        | 13             | 65            | 45            |
| VSS         | 68             | 10            | 11            |
| VSS         | 36             | 42            | 37            |
| WR          | 40             | 38            | 14            |
| XTAL1       | 67             | 11            | 36            |
| XTAL2       | 66             | 12            | 35            |

The Following pins are not bonded out in the 48-pin package:

P1.0 through P1.7, P0.0 through P0.3, P2.3, P2.4, P2.6, P2.7 CLKOUT, INST, NMI, TEST, T2CLK(P2.3), T2RST(P2.4).

## ARCHITECTURAL OVERVIEW

### 2.18. PACKAGE PIN-OUTS

Figures 2-32, 2-33, and 2-34 show the pin placements for the three packages. The mapping of pin numbers to functionality is different for each part so that the pin numbers can conform to JEDEC standards. The die in the PGA

package is mounted upside-down relative to the other packages, so the pin functions run clockwise instead of counter-clockwise. This was done to provide better thermal transfer properties in the PGA. Because of this mirror imaging, a PGA mounted on the bottom of the board can be used to prototype a PLCC design.

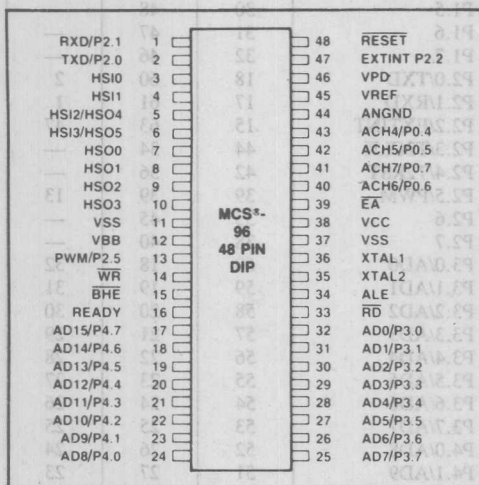


Figure 2-32. 48-Pin Package

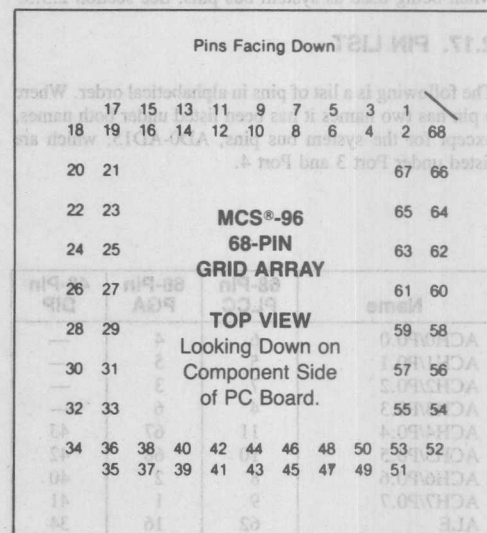


Figure 2-33. Pin Grid Array

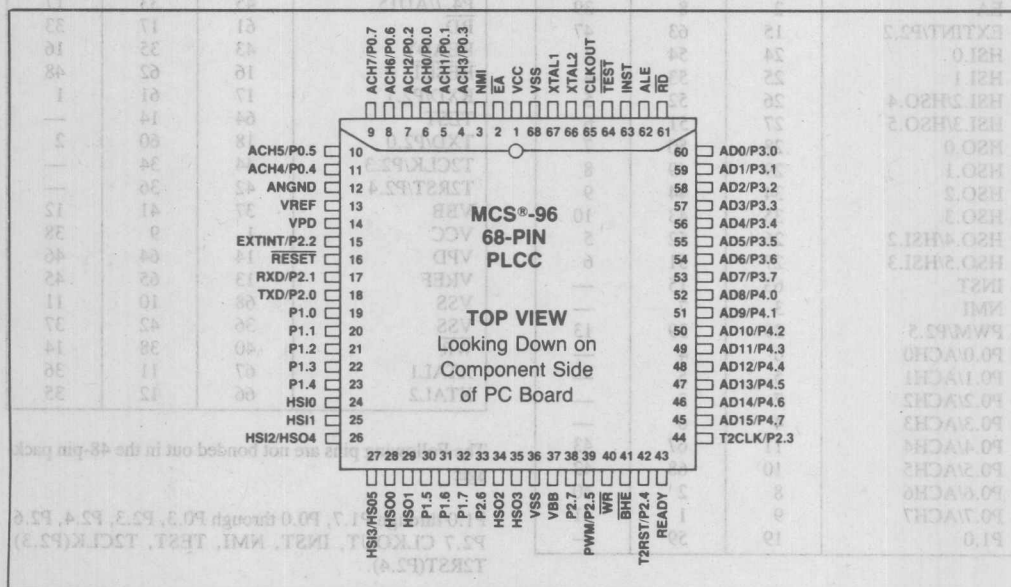


Figure 2-34. 68-Pin Package (PLCC — Top View)



---

**MCS®-96 Software  
Design Information**

---

**3**



## CHAPTER 3

# MCS®-96 SOFTWARE DESIGN INFORMATION

### 3.0. INTRODUCTION

This section provides information which will primarily interest those who must write programs to execute in the 8096. Several other sources of information are currently available which will also be of interest:

**MCS®-96 MACRO ASSEMBLER USER'S GUIDE**  
Order Number 122048-001

**MCS-96 UTILITIES USER'S GUIDE**  
Order Number 122049-001

**MCS-96 MACRO ASSEMBLER AND UTILITIES  
POCKET REFERENCE**  
Order Number 122050-001

Throughout this chapter short segments of code are used to illustrate the operation of the device. For these sections it has been assumed that a set of temporary registers have been predeclared. The names of these registers have been chosen as follows:

AX, BX, CX, and DX are 16 bit registers.

AL is the low byte of AX, AH is the high byte.

BL is the low byte of BX

CL is the low byte of CX

DL is the low byte of DX

These are the same as the names for the general data registers used in 8086. It is important to note, however, that in the 8096, these are not dedicated registers but merely the symbolic names assigned by the programmer to an eight byte region within onboard register file.

### 3.1. OPERAND TYPES

The MCS®-96 architecture provides support for a variety of data types which are likely to be useful in a control application. In the discussion of these operand types that follows, the names adopted by the PLM-96 programming language will be used where appropriate. To avoid confusion the name of an operand type will be capitalized. A "BYTE" is an unsigned eight bit variable; a "byte" is an eight bit unit of data of any type.

#### 3.1.1. Bytes

BYTES are unsigned 8-bit variables which can take on the values between 0 and 255. Arithmetic and relational operators can be applied to BYTE operands but the result must be interpreted in modulo 256 arithmetic. Logical operations on BYTES are applied bitwise. Bits within BYTES are labeled from 0 to 7 with 0 being the least significant bit. There are no alignment restrictions for BYTES so they may be placed anywhere in the MCS-96 address space.

#### 3.1.2. Words

WORDS are unsigned 16-bit variables which can take on the values between 0 and 65535. Arithmetic and relational

operators can be applied to WORD operands but the result must be interpreted modulo 65536. Logical operations on WORDS are applied bitwise. Bits within words are labeled from 0 to 15 with 0 being the least significant bit. WORDS must be aligned at even byte boundaries in the MCS-96 address space. The least significant byte of the WORD is in the even byte address and the most significant byte is in the next higher (odd) address. The address of a word is the address of its least significant byte. Word operations to odd addresses are not guaranteed to operate in a consistent manner.

#### 3.1.3. Short-Integers

SHORT-INTEGERs are 8-bit signed variables which can take on the values between -128 and +127. Arithmetic operations which generate results outside of the range of a SHORT-INTEGER will set the overflow indicators in the program status word. The actual numeric result returned will be the same as the equivalent operation on BYTE variables. There are no alignment restrictions on SHORT-INTEGERs so they may be placed anywhere in the MCS-96 address space.

#### 3.1.4. Integers

INTEGERs are 16-bit signed variables which can take on the values between -32,768 and 32,767. Arithmetic operations which generate results outside of the range of an INTEGER will set the overflow indicators in the program status word. The actual numeric result returned will be the same as the equivalent operation on WORD variables. INTEGERs conform to the same alignment and addressing rules as do WORDS.

#### 3.1.5. Bits

BITS are single-bit operands which can take on the Boolean values of true and false. In addition to the normal support for bits as components of BYTE and WORD operands, the 8096 provides for the direct testing of any bit in the internal register file. The MCS-96 architecture requires that bits be addressed as components of BYTES or WORDS, it does not support the direct addressing of bits that can occur in the MCS-51 architecture.

#### 3.1.6. Double-Words

DOUBLE-WORDs are unsigned 32-bit variables which can take on the values between 0 and 4,294,967,295. The MCS-96 architecture provides direct support for this operand type only for shifts and as the dividend in a 32 by 16 divide and the product of a 16 by 16 multiply. For these operations a DOUBLE-WORD variable must reside in the on-board register file of the 8096 and be aligned at an address which is evenly divisible by 4. A DOUBLE-WORD operand is addressed by the address of its least significant byte. DOUBLE-WORD operations which are not directly supported can be easily implemented with two WORD operations. For consistency with INTEL provided software the user should adopt the conventions for addressing DOUBLE-WORD operands which are discussed in section 3.5.

### 3.1.7. Long-Integers

LONG-INTEGERS are 32-bit signed variables which can take on the values between -2,147,483,648 and 2,147,483,647. The MCS-96 architecture provides direct support for this data type only for shifts and as the dividend in a 32 by 16 divide and the product of a 16 by 16 multiply.

LONG-INTEGERS can also be normalized. For these operations a LONG-INTEGER variable must reside in the onboard register file of the 8096 and be aligned at an

address which is evenly divisible by 4. A LONG-INTEGER is addressed by the address of its least significant byte.

LONG-INTEGER operations which are not directly supported can be easily implemented with two INTEGER operations. For consistency with Intel provided software, the user should adopt the conventions for addressing LONG operands which are discussed in section 3.5.

## 3.2. OPERAND ADDRESSING

Operands are accessed within the address space of the 8096 with one of six basic addressing modes. Some of the details of how these addressing modes work are hidden by the assembly language. If the programmer is to take full advantage of the architecture, it is important that these details be understood. This section will describe the addressing modes as they are handled by the hardware. At the end of this section the addressing modes will be de-

scribed as they are seen through the assembly language. The six basic addressing modes which will be described are termed register-direct, indirect, indirect with auto-increment, immediate, short-indexed, and long-indexed. Several other useful addressing operations can be achieved by combining these basic addressing modes with specific registers such as the ZERO register or the stack pointer.

### 3.2.1. Register-direct References

The register-direct mode is used to directly access a register from the 256 byte on-board register file. The register is selected by an 8-bit field within the instruction and

register address must conform to the alignment rules for the operand type. Depending on the instruction, up to three registers can take part in the calculation.

#### Examples

```
ADD    AX,BX,CX      ; AX := BX + CX
MUL    AX,BX          ; AX := AX * BX
INCB   CL             ; CL := CL + 1
```

### 3.2.2. Indirect References

The indirect mode is used to access an operand by placing its address in a WORD variable in the register file. The calculated address must conform to the alignment rules for the operand type. Note that the indirect address can refer to an operand anywhere within the address space of

the 8096, including the register file. The register which contains the indirect address is selected by an eight bit field within the instruction. An instruction can contain only one indirect reference and the remaining operands of the instruction (if any) must be register-direct references.

#### Examples

```
LD      AX,[AX]      ; AX := MEM _ WORD(AX)
ADDB    AL,BL,[CX]    ; AL := BL + MEM _ BYTE(CX)
POP     [AX]          ; MEM _ WORD(AX) := MEM _ WORD(SP); SP := SP + 2
```

### 3.2.3. Indirect with Auto-increment References

This addressing mode is the same as the indirect mode except that the WORD variable which contains the indirect address is incremented *after* it is used to address the operand. If the instruction operates on BYTES or SHORT-

INTEGERS the indirect address variable will be incremented by one, if the instruction operates on WORDS or INTEGERS the indirect address variable will be incremented by two.

#### Examples

```
LD      AX,[BX]++    ; AX := MEM _ WORD(BX); BX := BX + 2
ADDB    AL,BL,[CX]++ ; AL := AL + BL + MEM _ BYTE(CX); CX := CX + 1
PUSH    [AX]++       ; SP := SP - 2;
                     ; MEM _ WORD(SP) := MEM _ WORD(AX)
                     ; AX := AX + 2
```



### 3.2.4. Immediate References

This addressing mode allows an operand to be taken directly from a field in the instruction. For operations on BYTE or SHORT-INTEGER operands this field is eight bits wide, for operations on WORD or INTEGER oper-

ands the field is 16 bits wide. An instruction can contain only one immediate reference and the remaining operand(s) must be register-direct references.

#### Examples

```
ADD AX,#340 ; AX:=AX+340
PUSH #1234H ; SP:=SP-2; MEM _ WORD(SP):=1234H
DIBV AX,#10 ; AL:=AX/10; AH:=AX MOD 10
```

### 3.2.5. Short-indexed References

In this addressing mode an eight bit field in the instruction selects a WORD variable in the register file which is assumed to contain an address. A second eight bit field in the instruction stream is sign-extended and summed with the WORD variable to form the address of the operand which will take part in the calculation. Since the

eight bit field is sign-extended the effective address can be up to 128 bytes before the address in the WORD variable and up to 127 bytes after it. An instruction can contain only one short-indexed reference and the remaining operand(s) must be register-direct references.

#### Examples

```
LD AX,12[BX] ; AX:=MEM _ WORD(BX+12)
MULB AX,BL,3[CX] ; AX:=BL*MEM _ BYTE(CX+3)
```

### 3.2.6. Long-indexed References

This addressing mode is like the short-indexed mode except that a 16-bit field is taken from the instruction and added to the WORD variable to form the address of the

operand. No sign extension is necessary. An instruction can contain only one long-indexed reference and the remaining operand(s) must to register-direct references.

#### Examples

```
AND AX,BX,TABLE[CX] ; AX:=BX AND MEM _ WORD(TABLE+CX)
ST AX,TABLE[BX] ; MEM _ WORD(TABLE+BX):=AX
ADDB AL,BL,LOOKUP[CX] ; AL:=BL+MEM _ BYTE(LOOKUP+CX)
```

### 3.2.7. ZERO Register Addressing

The first two bytes in the register file are fixed at zero by the 8096 hardware. In addition to providing a fixed source of the constant zero for calculations and comparisons, this register can be used as the WORD variable in a long-

indexed reference. This combination of register selection and address mode allows any location in memory to be addressed directly.

#### Examples

```
ADD AX,1234[0] ; AX:=AX+MEM _ WORD(1234)
POP 5678[0] ; MEM _ WORD(5678):=MEM _ WORD(SP)
; SP:=SP+2
```

### 3.2.8. Stack Pointer Register Addressing

The system stack pointer in the 8096 can be accessed as register 18H of the internal register file. In addition to providing for convenient manipulation of the stack pointer, this also facilitates the accessing of operands in the stack. The top of the stack, for example, can be accessed by

using the stack pointer as the WORD variable in an indirect reference. In a similar fashion, the stack pointer can be used in the short-indexed mode to access data within the stack.

#### Examples

```
PUSH [SP] ; DUPLICATE TOP _ OF _ STACK
LD AX,2[SP] ; AX:=NEXT _ TO _ TOP
```

### 3.2.9. Assembly Language Addressing Modes

The 8096 assembly language simplifies the choice of addressing modes to be used in several respects:

**Direct Addressing.** The assembly language will choose between register-direct addressing and long-indexed with the ZERO register depending on where the operand is in memory. The user can simply refer to an operand by its symbolic name; if the operand is in the register file, a register-direct reference will be used, if the operand is elsewhere in memory, a long-indexed reference will be generated.

**Indexed Addressing.** The assembly language will choose between short and long indexing depending on the value of the index expression. If the value can be expressed in eight bits then short indexing will be used, if it cannot be expressed in eight bits then long indexing will be used.

The use of these features of the assembly language simplifies the programming task and should be used wherever possible.

### 3.3 PROGRAM STATUS WORD

The program status word (PSW) is a collection of Boolean flags which retain information concerning the state of the user's program. The format of the PSW is shown in figure 3-1. The information in the PSW can be broken down into

|    |    |    |    |    |    |    |    |                      |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|----------------------|----|----|----|----|----|----|----|
| 15 | 14 | 13 | 12 | 11 | 10 | 09 | 08 | 07                   | 06 | 05 | 04 | 03 | 02 | 01 | 00 |
| Z  | N  | V  | VT | C  | —  | I  | ST | <Interrupt Mask Reg> |    |    |    |    |    |    |    |

Figure 3-1. PSW Register

#### 3.3.1. Interrupt Flags

The lower eight bits of the PSW are used to individually mask the various sources of interrupt to the 8096. A logical '1' in these bit positions enables the servicing of the corresponding interrupt. These mask bits can be accessed as an eight bit byte (INT\_MASK — address 8) in the on-board register file. Bit 9 in the PSW is the global interrupt enable. If this bit is cleared then all interrupts will be locked out except for the Non Maskable Interrupt (NMI). Note that the various interrupts are collected in the INT\_PENDING register even if they are locked out. Execution of the corresponding service routines will proceed according to their priority when they become enabled. Further information on the interrupt structure of the 8096 can be found in sections 2.5 and 3.6.

#### 3.3.2. Condition Flags

The remaining bits in the PSW are set as side effects of instruction execution and can be tested by the conditional jump instructions.

**Z.** The Z (Zero) flag is set to indicate that the operation generated a result equal to zero. For the add-with-carry (ADDC) and subtract-with-borrow (SUBC) operations the Z flag is cleared if the result is non-zero but is never set. These two instructions are normally used in conjunction with the ADD and SUB instructions to perform multiple precision arithmetic. The operation of the Z flag for these instructions leaves it indicating the proper result for the entire multiple precision calculation.

**N.** The N (Negative) flag is set to indicate that the op-

eration generated a negative result. Note that the N flag will be set to the algebraically correct state even if the calculation overflows.

**V.** The V (oVerflow) flag is set to indicate that the operation generated a result which is outside the range that can be expressed in the destination data type.

**VT.** The VT (oVerflow Trap) flag is set whenever the V flag is set but can only be cleared by an instruction which explicitly operates on it such as the CLRVT or JVT instructions. The operation of the VT flag allows for the testing for a possible overflow condition at the end of a sequence of related arithmetic operations. This is normally more efficient than testing the V flag after each instruction.

**C.** The C (Carry) flag is set to indicate the state of the arithmetic carry from the most significant bit of the ALU for an arithmetic operation or the state of the last bit shifted out of the operand for a shift. Arithmetic Borrow after a subtract operation is the complement of the C flag (i.e. if the operation generated a borrow then C=0).

**ST.** The ST (STicky bit) set to indicate that during a right shift a 1 has been shifted first into the C flag and then been shifted out. The ST flag is undefined after a multiply operation. The ST flag can be used along with the C flag to control rounding after a right shift. Consider multiplying two eight bit quantities and then scaling the result down to 12 bits:

```
MULUB  AX,CL,DL ; AX=CL*DL
SHR     AX,#4    ; Shift right 4 places
```



Table 3-1. Instruction Summary

| Mnemonic        | Operands | Operation (Note 1)   | Flags |   |   |   |    |    | Notes |
|-----------------|----------|--|-------|---|---|---|----|----|-------|
|                 |          |  | Z     | N | C | V | VT | ST |       |
| ADD/ADDB        | 2        | $D \leftarrow D + A$   | ✓     | ✓ | ✓ | ✓ | ↑  | —  |       |
| ADD/ADDB        | 3        | $D \leftarrow B + A$   | ✓     | ✓ | ✓ | ✓ | ↑  | —  |       |
| ADDC/ADDCB      | 2        | $D \leftarrow D + A + C$   | ↓     | ✓ | ✓ | ✓ | ↑  | —  |       |
| SUB/SUBB        | 2        | $D \leftarrow D - A$   | ✓     | ✓ | ✓ | ✓ | ↑  | —  |       |
| SUB/SUBB        | 3        | $D \leftarrow B - A$   | ✓     | ✓ | ✓ | ✓ | ↑  | —  |       |
| SUBC/SUBCB      | 2        | $D \leftarrow D - A + C - 1$   | ↓     | ✓ | ✓ | ✓ | ↑  | —  |       |
| CMP/CMPB        | 2        | $D - A$  | ✓     | ✓ | ✓ | ✓ | ↑  | —  |       |
| MUL/MULU        | 2        | $D, D + 2 \leftarrow D * A$  | —     | — | — | — | —  | ?  | 2     |
| MUL/MULU        | 3        | $D, D + 2 \leftarrow B * A$  | —     | — | — | — | —  | ?  | 2     |
| MULB/MULUB      | 2        | $D, D + 1 \leftarrow D * A$  | —     | — | — | — | —  | ?  | 3     |
| MULB/MULUB      | 3        | $D, D + 1 \leftarrow B * A$  | —     | — | — | — | —  | ?  | 3     |
| DIVU            | 2        | $D \leftarrow (D, D + 2)/A; D + 2 \leftarrow \text{remainder}$                                       | —     | — | — | ✓ | ↑  | —  | 2     |
| DIVUB           | 2        | $D \leftarrow (D, D + 1)/A; D + 1 \leftarrow \text{remainder}$                                       | —     | — | — | ✓ | ↑  | —  | 3     |
| DIV             | 2        | $D \leftarrow (D, D + 2)/A; D + 2 \leftarrow \text{remainder}$                                       | —     | — | — | ? | ↑  | —  |       |
| DIVB            | 2        | $D \leftarrow (D, D + 1)/A; D + 1 \leftarrow \text{remainder}$                                       | —     | — | — | ? | ↑  | —  |       |
| AND/ANDB        | 2        | $D \leftarrow D \text{ and } A$  | ✓     | ✓ | 0 | 0 | —  | —  |       |
| AND/ANDB        | 3        | $D \leftarrow B \text{ and } A$  | ✓     | ✓ | 0 | 0 | —  | —  |       |
| OR/ORB          | 2        | $D \leftarrow D \text{ or } A$   | ✓     | ✓ | 0 | 0 | —  | —  |       |
| XOR/XORB        | 2        | $D \leftarrow D \text{ (excl. or) } A$   | ✓     | ✓ | 0 | 0 | —  | —  |       |
| LD/LDB          | 2        | $D \leftarrow A$   | —     | — | — | — | —  | —  |       |
| ST/STB          | 2        | $A \leftarrow D$   | —     | — | — | — | —  | —  |       |
| LDBSE           | 2        | $D \leftarrow A; D + 1 \leftarrow \text{SIGN}(A)$  | —     | — | — | — | —  | —  | 3, 4  |
| LDBZE           | 2        | $D \leftarrow A; D + 1 \leftarrow 0$   | —     | — | — | — | —  | —  | 3, 4  |
| PUSH            | 1        | $SP \leftarrow SP - 2; (SP) \leftarrow A$  | —     | — | — | — | —  | —  |       |
| POP             | 1        | $A \leftarrow (SP); SP \leftarrow SP + 2$  | —     | — | — | — | —  | —  |       |
| PUSHF           | 0        | $SP \leftarrow SP - 2; (SP) \leftarrow \text{PSW};$<br>$\text{PSW} \leftarrow 0000H; I \leftarrow 0$ | 0     | 0 | 0 | 0 | 0  | 0  |       |
| POPF            | 0        | $\text{PSW} \leftarrow (SP); SP \leftarrow SP + 2; I \leftarrow \text{PSW}$                          | ✓     | ✓ | ✓ | ✓ | ✓  | ✓  |       |
| SJMP            | 1        | $PC \leftarrow PC + 11\text{-bit offset}$  | —     | — | — | — | —  | —  | 5     |
| LJMP            | 1        | $PC \leftarrow PC + 16\text{-bit offset}$  | —     | — | — | — | —  | —  | 5     |
| BR [indirect]   | 1        | $PC \leftarrow (A)$  | —     | — | — | — | —  | —  |       |
| SCALL           | 1        | $SP \leftarrow SP - 2; (SP) \leftarrow PC;$<br>$PC \leftarrow PC + 11\text{-bit offset}$             | —     | — | — | — | —  | —  | 5     |
| LCALL           | 1        | $SP \leftarrow SP - 2; (SP) \leftarrow PC;$<br>$PC \leftarrow PC + 16\text{-bit offset}$             | —     | — | — | — | —  | —  | 5     |
| RET             | 0        | $PC \leftarrow (SP); SP \leftarrow SP + 2$   | —     | — | — | — | —  | —  |       |
| J (conditional) | 1        | $PC \leftarrow PC + 8\text{-bit offset (if taken)}$  | —     | — | — | — | —  | —  | 5     |
| JC              | 1        | Jump if C = 1  | —     | — | — | — | —  | —  | 5     |
| JNC             | 1        | Jump if C = 0  | —     | — | — | — | —  | —  | 5     |
| JE              | 1        | Jump if Z = 1  | —     | — | — | — | —  | —  | 5     |

**NOTES:**

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
2. D, D + 2 are consecutive WORDS in memory; D is DOUBLE-WORD aligned.
3. D, D + 1 are consecutive BYTES in memory; D is WORD aligned.
4. Changes a byte to a word.
5. Offset is a 2's complement number.



Table 3-2. Instruction Summary

| Mnemonic         | Operands | Operation (Note 1)                                 | Flags |   |   |   |    |    | Notes |
|------------------|----------|--|-------|---|---|---|----|----|-------|
|                  |          |  | Z     | N | C | V | VT | ST |       |
| JNE              | 1        | Jump if Z = 0                                      | —     | — | — | — | —  | —  | 5     |
| JGE              | 1        | Jump if N = 0                                      | —     | — | — | — | —  | —  | 5     |
| JLT              | 1        | Jump if N = 1                                      | —     | — | — | — | —  | —  | 5     |
| JGT              | 1        | Jump if N = 0 and Z = 0                            | —     | — | — | — | —  | —  | 5     |
| JLE              | 1        | Jump if N = 1 or Z = 1                             | —     | — | — | — | —  | —  | 5     |
| JH               | 1        | Jump if C = 1 and Z = 0                            | —     | — | — | — | —  | —  | 5     |
| JNH              | 1        | Jump if C = 0 or Z = 1                             | —     | — | — | — | —  | —  | 5     |
| JV               | 1        | Jump if V = 1                                      | —     | — | — | — | —  | —  | 5     |
| JNV              | 1        | Jump if V = 0                                      | —     | — | — | — | —  | —  | 5     |
| JVT              | 1        | Jump if VT = 1; Clear VT                           | —     | — | — | — | 0  | —  | 5     |
| JNVT             | 1        | Jump if VT = 0; Clear VT                           | —     | — | — | — | 0  | —  | 5     |
| JST              | 1        | Jump if ST = 1                                     | —     | — | — | — | —  | —  | 5     |
| JNST             | 1        | Jump if ST = 0                                     | —     | — | — | — | —  | —  | 5     |
| JBS              | 3        | Jump if Specified Bit = 1                          | —     | — | — | — | —  | —  | 5, 6  |
| JBC              | 3        | Jump if Specified Bit = 0                          | —     | — | — | — | —  | —  | 5, 6  |
| DJNZ             | 1        | D ← D - 1; if D ≠ 0 then<br>PC ← PC + 8-bit offset | —     | — | — | — | —  | —  | 5     |
| DEC/DECB         | 1        | D ← D - 1  | ✓     | ✓ | ✓ | ✓ | ↑  | —  |       |
| NEG/NEGB         | 1        | D ← 0 - D  | ✓     | ✓ | ✓ | ✓ | ↑  | —  |       |
| INC/INCB         | 1        | D ← D + 1  | ✓     | ✓ | ✓ | ✓ | ↑  | —  |       |
| EXT              | 1        | D ← D; D + 2 ← Sign(D)                             | ✓     | ✓ | 0 | 0 | —  | —  | 2     |
| EXTB             | 1        | D ← D; D + 1 ← Sign(D)                             | ✓     | ✓ | 0 | 0 | —  | —  | 3     |
| NOT/NOTB         | 1        | D ← Logical Not (D)                                | ✓     | ✓ | 0 | 0 | —  | —  |       |
| CLR/CLRB         | 1        | D ← 0  | 1     | 0 | 0 | 0 | —  | —  |       |
| SHL/SHLB/SHLL    | 2        | C ← msb ———— lsb ← 0                               | ✓     | ? | ✓ | ✓ | ↑  | —  | 7     |
| SHR/SHRB/SHRL    | 2        | 0 → msb ———— lsb → C                               | ✓     | ? | ✓ | 0 | —  | ✓  | 7     |
| SHRA/SHRAB/SHRAL | 2        | msb → msb ———— lsb → C                             | ✓     | ✓ | ✓ | 0 | —  | ✓  | 7     |
| SETC             | 0        | C ← 1  | —     | — | 1 | — | —  | —  |       |
| CLRC             | 0        | C ← 0  | —     | — | 0 | — | —  | —  |       |
| CLRVT            | 0        | VT ← 0   | —     | — | — | — | 0  | —  |       |
| RST              | 0        | PC ← 2080H   | 0     | 0 | 0 | 0 | 0  | 0  | 8     |
| DI               | 0        | Disable All Interrupts (I ← 0)                     | —     | — | — | — | —  | —  |       |
| EI               | 0        | Enable All Interrupts (I ← 1)                      | —     | — | — | — | —  | —  |       |
| NOP              | 0        | PC ← PC + 1  | —     | — | — | — | —  | —  |       |
| SKIP             | 0        | PC ← PC + 2  | —     | — | — | — | —  | —  |       |
| NORML            | 2        | Left shift till msb = 1; D ← shift count           | ✓     | ? | 0 | — | —  | —  | 7     |
| TRAP             | 0        | SP ← SP - 2; (SP) ← PC<br>PC ← (2010H)             | —     | — | — | — | —  | —  | 9     |

**NOTES:**

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
5. Offset is a 2's complement number.
6. Specified bit is one of the 2048 bits in the register file.
7. The "L" (Long) suffix indicates double-word operation.
8. Initiates a Reset by pulling RESET low. Software should re-initialize all the necessary registers with code starting at 2080H.
9. The assembler will not accept this mnemonic.

Table 3-3. Opcode and State Time Listing

| MNEMONIC                | OPERANDS | DIRECT |       |             | IMMEDIATE |       |             | INDIRECT <sup>Ⓢ</sup> |       |                          |           |                          | INDEXED <sup>Ⓢ</sup> |       |                          |       |                          |
|-------------------------|----------|--------|-------|-------------|-----------|-------|-------------|-----------------------|-------|--------------------------|-----------|--------------------------|----------------------|-------|--------------------------|-------|--------------------------|
|                         |          | OPCODE | BYTES | STATE TIMES | OPCODE    | BYTES | STATE TIMES | NORMAL                |       |                          | AUTO-INC. |                          | SHORT                |       |                          | LONG  |                          |
|                         |          |        |       |             |           |       |             | OPCODE                | BYTES | STATE <sup>①</sup> TIMES | BYTES     | STATE <sup>①</sup> TIMES | OPCODE               | BYTES | STATE <sup>①</sup> TIMES | BYTES | STATE <sup>①</sup> TIMES |
|                         |          |        |       |             |           |       |             |                       |       |                          |           |                          |                      |       |                          |       |                          |
| ARITHMETIC INSTRUCTIONS |          |        |       |             |           |       |             |                       |       |                          |           |                          |                      |       |                          |       |                          |
| ADD                     | 2        | 64     | 3     | 4           | 65        | 4     | 5           | 66                    | 3     | 6/11                     | 3         | 7/12                     | 67                   | 4     | 6/11                     | 5     | 7/12                     |
| ADD                     | 3        | 44     | 4     | 5           | 45        | 5     | 6           | 46                    | 4     | 7/12                     | 4         | 8/13                     | 47                   | 5     | 7/12                     | 6     | 8/13                     |
| ADDB                    | 2        | 74     | 3     | 4           | 75        | 3     | 4           | 76                    | 3     | 6/11                     | 3         | 7/12                     | 77                   | 4     | 6/11                     | 5     | 7/12                     |
| ADDB                    | 3        | 54     | 4     | 5           | 55        | 4     | 5           | 56                    | 4     | 7/12                     | 4         | 8/13                     | 57                   | 5     | 7/12                     | 6     | 8/13                     |
| ADDC                    | 2        | A4     | 3     | 4           | A5        | 4     | 5           | A6                    | 3     | 6/11                     | 3         | 7/12                     | A7                   | 4     | 6/11                     | 5     | 7/12                     |
| ADDCB                   | 2        | B4     | 3     | 4           | B5        | 3     | 4           | B6                    | 3     | 6/11                     | 3         | 7/12                     | B7                   | 4     | 6/11                     | 5     | 7/12                     |
| SUB                     | 2        | 68     | 3     | 4           | 69        | 4     | 5           | 6A                    | 3     | 6/11                     | 3         | 7/12                     | 6B                   | 4     | 6/11                     | 5     | 7/12                     |
| SUB                     | 3        | 48     | 4     | 5           | 49        | 5     | 6           | 4A                    | 4     | 7/12                     | 4         | 8/13                     | 4B                   | 5     | 7/12                     | 6     | 8/13                     |
| SUBB                    | 2        | 78     | 3     | 4           | 79        | 3     | 4           | 7A                    | 3     | 6/11                     | 3         | 7/12                     | 7B                   | 4     | 6/11                     | 5     | 7/12                     |
| SUBB                    | 3        | 58     | 4     | 5           | 59        | 4     | 5           | 5A                    | 4     | 7/12                     | 4         | 8/13                     | 5B                   | 5     | 7/12                     | 6     | 8/13                     |
| SUBC                    | 2        | A8     | 3     | 4           | A9        | 4     | 5           | AA                    | 3     | 6/11                     | 3         | 7/12                     | AB                   | 4     | 6/11                     | 5     | 7/12                     |
| SUBCB                   | 2        | B8     | 3     | 4           | B9        | 3     | 4           | BA                    | 3     | 6/11                     | 3         | 7/12                     | BB                   | 4     | 6/11                     | 5     | 7/12                     |
| CMP                     | 2        | 88     | 3     | 4           | 89        | 4     | 5           | 8A                    | 3     | 6/11                     | 3         | 7/12                     | 8B                   | 4     | 6/11                     | 5     | 7/12                     |
| CMPB                    | 2        | 98     | 3     | 4           | 99        | 3     | 4           | 9A                    | 3     | 6/11                     | 3         | 7/12                     | 9B                   | 4     | 6/11                     | 5     | 7/12                     |
|                         |          |        |       |             |           |       |             |                       |       |                          |           |                          |                      |       |                          |       |                          |
| MULU                    | 2        | 6C     | 3     | 25          | 6D        | 4     | 26          | 6E                    | 3     | 27/32                    | 3         | 28/33                    | 6F                   | 4     | 27/32                    | 5     | 28/33                    |
| MULU                    | 3        | 4C     | 4     | 26          | 4D        | 5     | 27          | 4E                    | 4     | 28/33                    | 4         | 29/34                    | 4F                   | 5     | 28/33                    | 6     | 29/34                    |
| MULUB                   | 2        | 7C     | 3     | 17          | 7D        | 3     | 17          | 7E                    | 3     | 19/24                    | 3         | 20/25                    | 7F                   | 4     | 19/24                    | 5     | 20/25                    |
| MULUB                   | 3        | 5C     | 4     | 18          | 5D        | 4     | 18          | 5E                    | 4     | 20/25                    | 4         | 21/26                    | 5F                   | 5     | 20/25                    | 6     | 21/26                    |
| MUL                     | 2        | Ⓢ      | 4     | 29          | Ⓢ         | 5     | 30          | Ⓢ                     | 4     | 31/36                    | 4         | 32/37                    | Ⓢ                    | 5     | 31/36                    | 6     | 32/37                    |
| MUL                     | 3        | Ⓢ      | 5     | 30          | Ⓢ         | 6     | 31          | Ⓢ                     | 5     | 32/37                    | 5         | 33/38                    | Ⓢ                    | 6     | 32/37                    | 7     | 33/38                    |
| MULB                    | 2        | Ⓢ      | 4     | 21          | Ⓢ         | 4     | 21          | Ⓢ                     | 4     | 23/28                    | 4         | 24/29                    | Ⓢ                    | 5     | 23/28                    | 6     | 24/29                    |
| MULB                    | 3        | Ⓢ      | 5     | 22          | Ⓢ         | 5     | 22          | Ⓢ                     | 5     | 24/29                    | 5         | 25/30                    | Ⓢ                    | 6     | 24/29                    | 7     | 25/30                    |
| DIVU                    | 2        | 8C     | 3     | 25          | 8D        | 4     | 26          | 8E                    | 3     | 28/32                    | 3         | 29/33                    | 8F                   | 4     | 28/32                    | 5     | 29/33                    |
| DIVUB                   | 2        | 9C     | 3     | 17          | 9D        | 3     | 17          | 9E                    | 3     | 20/24                    | 3         | 21/25                    | 9F                   | 4     | 20/24                    | 5     | 21/25                    |
| DIV                     | 2        | Ⓢ      | 4     | 29          | Ⓢ         | 5     | 30          | Ⓢ                     | 4     | 32/36                    | 4         | 33/37                    | Ⓢ                    | 5     | 32/36                    | 6     | 33/37                    |
| DIVB                    | 2        | Ⓢ      | 4     | 21          | Ⓢ         | 4     | 21          | Ⓢ                     | 4     | 24/28                    | 4         | 25/29                    | Ⓢ                    | 5     | 24/28                    | 6     | 25/29                    |

## Notes:

② Long indexed and Indirect + instructions have identical opcodes with Short indexed and Indirect modes, respectively. The second byte of instructions using any indirect or indexed addressing mode specifies the exact mode used. If the second byte is even, use Indirect or Short Indexed. If it is odd, use Indirect + or Long indexed. In all cases the second byte of the instruction always specifies an even (word) location for the address referenced.

① Number of state times shown for internal/external operands.

② The opcodes for signed multiply and divide are the opcodes for the unsigned functions with an "FE" appended as a prefix.

Table 3-3. Continued

| MNEMONIC                          | OPERANDS           | DIRECT |       |             | IMMEDIATE |                   |                    | INDIRECT <sup>②</sup> |       |                          |        | INDEXED <sup>③</sup> |                          |   |       |   |       |
|-----------------------------------|--------------------|--------|-------|-------------|-----------|-------------------|--------------------|-----------------------|-------|--------------------------|--------|----------------------|--------------------------|---|-------|---|-------|
|                                   |                    | NORMAL |       | AUTO-INC.   |           | SHORT             |                    | LONG                  |       |                          |        |                      |                          |   |       |   |       |
|                                   |                    | OPCODE | BYTES | STATE TIMES | OPCODE    | BYTES             | STATE TIMES        | OPCODE                | BYTES | STATE <sup>①</sup> TIMES | OPCODE | BYTES                | STATE <sup>①</sup> TIMES |   |       |   |       |
| LOGICAL INSTRUCTIONS              |                    |        |       |             |           |                   |                    |                       |       |                          |        |                      |                          |   |       |   |       |
| AND                               | 2                  | 60     | 3     | 4           | 61        | 4                 | 5                  | 62                    | 3     | 6/11                     | 3      | 7/12                 | 63                       | 4 | 6/11  | 5 | 7/12  |
| AND                               | 3                  | 40     | 4     | 5           | 41        | 5                 | 6                  | 42                    | 4     | 7/12                     | 4      | 8/13                 | 43                       | 5 | 7/12  | 6 | 8/13  |
| ANDB                              | 2                  | 70     | 3     | 4           | 71        | 3                 | 4                  | 72                    | 3     | 6/11                     | 3      | 7/12                 | 73                       | 4 | 6/11  | 5 | 7/12  |
| ANDB                              | 3                  | 50     | 4     | 5           | 51        | 4                 | 5                  | 52                    | 4     | 7/12                     | 4      | 8/13                 | 53                       | 5 | 7/12  | 6 | 8/13  |
| OR                                | 2                  | 80     | 3     | 4           | 81        | 4                 | 5                  | 82                    | 3     | 6/11                     | 3      | 7/12                 | 83                       | 4 | 6/11  | 5 | 7/12  |
| ORB                               | 2                  | 90     | 3     | 4           | 91        | 3                 | 4                  | 92                    | 3     | 6/11                     | 3      | 7/12                 | 93                       | 4 | 6/11  | 5 | 7/12  |
| XOR                               | 2                  | 84     | 3     | 4           | 85        | 4                 | 5                  | 86                    | 3     | 6/11                     | 3      | 7/12                 | 87                       | 4 | 6/11  | 5 | 7/12  |
| XORB                              | 2                  | 94     | 3     | 4           | 95        | 3                 | 4                  | 96                    | 3     | 6/11                     | 3      | 7/12                 | 97                       | 4 | 6/11  | 5 | 7/12  |
| DATA TRANSFER INSTRUCTIONS        |                    |        |       |             |           |                   |                    |                       |       |                          |        |                      |                          |   |       |   |       |
| LD                                | 2                  | A0     | 3     | 4           | A1        | 4                 | 5                  | A2                    | 3     | 6/11                     | 3      | 7/12                 | A3                       | 4 | 6/11  | 5 | 7/12  |
| LDB                               | 2                  | B0     | 3     | 4           | B1        | 3                 | 4                  | B2                    | 3     | 6/11                     | 3      | 7/12                 | B3                       | 4 | 6/11  | 5 | 7/12  |
| ST                                | 2                  | C0     | 3     | 4           | —         | —                 | —                  | C2                    | 3     | 7/11                     | 3      | 8/12                 | C3                       | 4 | 7/11  | 5 | 8/12  |
| STB                               | 2                  | C4     | 3     | 4           | —         | —                 | —                  | C6                    | 3     | 7/11                     | 3      | 8/12                 | C7                       | 4 | 7/11  | 5 | 8/12  |
| LDBSE                             | 2                  | BC     | 3     | 4           | BD        | 3                 | 4                  | BE                    | 3     | 6/11                     | 3      | 7/12                 | BF                       | 4 | 6/11  | 5 | 7/12  |
| LDBZE                             | 2                  | AC     | 3     | 4           | AD        | 3                 | 4                  | AE                    | 3     | 6/11                     | 3      | 7/12                 | AF                       | 4 | 6/11  | 5 | 7/12  |
| STACK OPERATIONS (internal stack) |                    |        |       |             |           |                   |                    |                       |       |                          |        |                      |                          |   |       |   |       |
| PUSH                              | 1                  | C8     | 2     | 8           | C9        | 3                 | 8                  | CA                    | 2     | 11/15                    | 2      | 12/16                | CB                       | 3 | 11/15 | 4 | 12/16 |
| POP                               | 1                  | CC     | 2     | 12          | —         | —                 | —                  | CE                    | 2     | 14/18                    | 2      | 14/18                | CF                       | 3 | 14/18 | 4 | 14/18 |
| PUSHF                             | 0                  | F2     | 1     | 8           |           |                   |                    |                       |       |                          |        |                      |                          |   |       |   |       |
| POPF                              | 0                  | F3     | 1     | 9           |           |                   |                    |                       |       |                          |        |                      |                          |   |       |   |       |
| STACK OPERATIONS (external stack) |                    |        |       |             |           |                   |                    |                       |       |                          |        |                      |                          |   |       |   |       |
| PUSH                              | 1                  | C8     | 2     | 12          | C9        | 3                 | 12                 | CA                    | 2     | 15/19                    | 2      | 16/20                | CB                       | 3 | 15/19 | 4 | 16/20 |
| POP                               | 1                  | CC     | 2     | 14          | —         | —                 | —                  | CE                    | 2     | 16/20                    | 2      | 16/20                | CF                       | 3 | 16/20 | 4 | 16/20 |
| PUSHF                             | 0                  | F2     | 1     | 12          |           |                   |                    |                       |       |                          |        |                      |                          |   |       |   |       |
| POPF                              | 0                  | F3     | 1     | 13          |           |                   |                    |                       |       |                          |        |                      |                          |   |       |   |       |
| JUMPS AND CALLS                   |                    |        |       |             |           |                   |                    |                       |       |                          |        |                      |                          |   |       |   |       |
| MNEMONIC                          | OPCODE             | BYTES  |       | STATES      |           | MNEMONIC          | OPCODE             | BYTES                 |       | STATES                   |        |                      |                          |   |       |   |       |
| LJMP                              | E7                 | 3      |       | 8           |           | LCALL             | EF                 | 3                     |       | 13/16 <sup>⑤</sup>       |        |                      |                          |   |       |   |       |
| SJMP                              | 20-27 <sup>④</sup> | 2      |       | 8           |           | SCALL             | 28-2F <sup>④</sup> | 2                     |       | 13/16 <sup>⑤</sup>       |        |                      |                          |   |       |   |       |
| BR[ ]                             | E3                 | 2      |       | 8           |           | RET               | F0                 | 1                     |       | 12/16 <sup>⑤</sup>       |        |                      |                          |   |       |   |       |
| Notes:                            |                    |        |       |             |           | TRAP <sup>③</sup> |                    | F7                    |       | 1                        |        |                      |                          |   |       |   |       |

## Notes:

① Number of state times shown for internal/external operands.

② The assembler does not accept this mnemonic.

③ The least significant 3 bits of the opcode are concatenated with the following 8 bits to form an 11-bit, 2's complement, offset for the relative call or jump.

④ State times for stack located internal/external.

Table 3-4. CONDITIONAL JUMPS

All conditional jumps are 2 byte instructions. They require 8 state times if the jump is taken, 4 if it is not.

| MNEMONIC | OPCODE | MNEMONIC | OPCODE | MNEMONIC | OPCODE | MNEMONIC | OPCODE |
|----------|--------|----------|--------|----------|--------|----------|--------|
| JC       | DB     | JE       | DF     | JGE      | D6     | JGT      | D2     |
| JNC      | D3     | JNE      | D7     | JLT      | DE     | JLE      | DA     |
| JH       | D9     | JV       | DD     | JVT      | DC     | JST      | D8     |
| JNH      | D1     | JNV      | D5     | JNVT     | D4     | JNST     | D0     |

## JUMP ON BIT CLEAR OR BIT SET

These instructions are 3-byte instructions. They require 9 state times if the jump is taken, 5 if it is not.

|          |    | BIT NUMBER |    |    |    |    |    |    |
|----------|----|------------|----|----|----|----|----|----|
| MNEMONIC | 0  | 1          | 2  | 3  | 4  | 5  | 6  | 7  |
| JBC      | 30 | 31         | 32 | 33 | 34 | 35 | 36 | 37 |
| JBS      | 38 | 39         | 3A | 3B | 3C | 3D | 3E | 3F |

## LOOP CONTROL

|      |            |          |                                   |
|------|------------|----------|-----------------------------------|
| DJNZ | OPCODE EO; | 3 BYTES; | 5/9 STATE TIMES (NOT TAKEN/TAKEN) |
|------|------------|----------|-----------------------------------|

## SINGLE REGISTER INSTRUCTIONS

| MNEMONIC | OPCODE | BYTES | STATES | MNEMONIC | OPCODE | BYTES | STATES |
|----------|--------|-------|--------|----------|--------|-------|--------|
| DEC      | 05     | 2     | 4      | EXT      | 06     | 2     | 4      |
| DECB     | 15     | 2     | 4      | EXTB     | 16     | 2     | 4      |
| NEG      | 03     | 2     | 4      | NOT      | 02     | 2     | 4      |
| NEGB     | 13     | 2     | 4      | NOTB     | 12     | 2     | 4      |
| INC      | 07     | 2     | 4      | CLR      | 01     | 2     | 4      |
| INCB     | 17     | 2     | 4      | CLRB     | 11     | 2     | 4      |

## SHIFT INSTRUCTIONS

| INSTR    |    | WORD |          | INSTR |   | BYTE     |    | INSTR |          | DBL WD |   | STATE TIMES                  |
|----------|----|------|----------|-------|---|----------|----|-------|----------|--------|---|------------------------------|
| MNEMONIC | OP | B    | MNEMONIC | OP    | B | MNEMONIC | OP | B     | MNEMONIC | OP     | B |                              |
| SHL      | 09 | 3    | SHLB     | 19    | 3 | SHLL     | 0D | 3     |          |        |   | 7 + 1 PER SHIFT <sup>⑦</sup> |
| SHR      | 08 | 3    | SHRB     | 18    | 3 | SHRL     | 0C | 3     |          |        |   | 7 + 1 PER SHIFT <sup>⑦</sup> |
| SHRA     | 0A | 3    | SHRAB    | 1A    | 3 | SHRAL    | 0E | 3     |          |        |   | 7 + 1 PER SHIFT <sup>⑦</sup> |

## SPECIAL CONTROL INSTRUCTIONS

| MNEMONIC | OPCODE | BYTES | STATES | MNEMONIC | OPCODE | BYTES | STATES |
|----------|--------|-------|--------|----------|--------|-------|--------|
| SETC     | F9     | 1     | 4      | DI       | FA     | 1     | 4      |
| CLRC     | F8     | 1     | 4      | EI       | FB     | 1     | 4      |
| CLRVT    | FC     | 1     | 4      | NOP      | FD     | 1     | 4      |
| RST      | FF     | 1     | 16     | SKIP     | 00     | 2     | 4      |

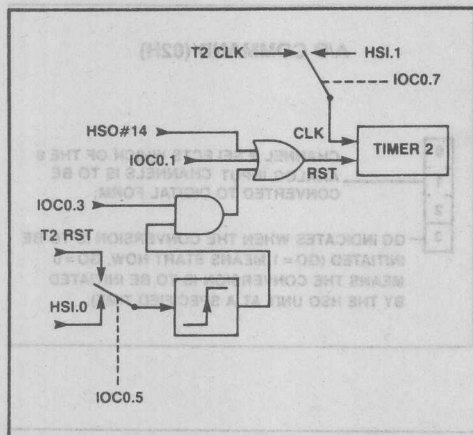
## NORMALIZE

|       |    |   |                  |
|-------|----|---|------------------|
| NORML | OF | 3 | 11 + 1 PER SHIFT |
|-------|----|---|------------------|

## Notes:

- ⑥ This instruction takes 2 states to pull RST low, then holds it low for 2 states to initiate a reset. The reset takes 12 states, at which time the program restarts at location 2080H.
- ⑦ Execution will take at least 8 states, even for 0 shift.

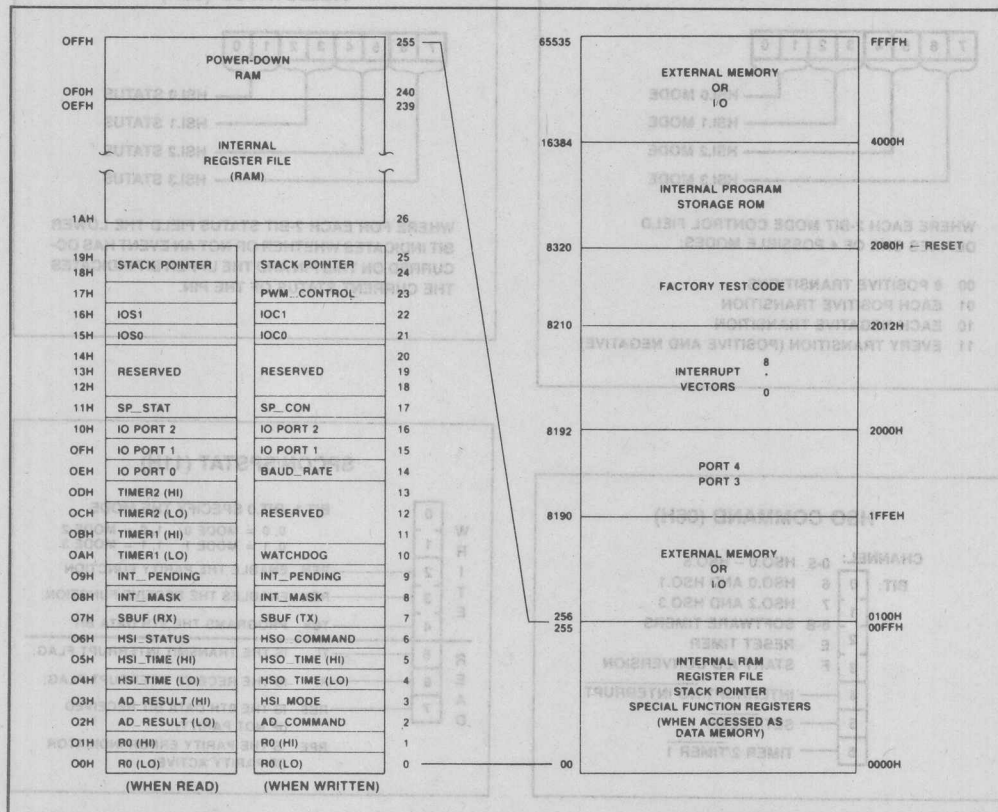




Timer 2 Clock and Reset Options

| Port | Function            | Alternate Function           | Controlled by |
|------|---------------------|------------------------------|---------------|
| P2.0 | output              | TXD (serial port transmit)   | IOC1.5        |
| P2.1 | input               | RXD (serial port receive)    | N/A           |
| P2.2 | input               | EXTINT (external interrupt)  | IOC1.1        |
| P2.3 | input               | T2CLK (Timer 2 input)        | IOC0.7        |
| P2.4 | input               | T2RST (Timer 2 reset)        | IOC0.5        |
| P2.5 | output              | PWM (pulse-width modulation) | IOC1.0        |
| P2.6 | quasi-bidirectional |                              |               |
| P2.7 | quasi-bidirectional |                              |               |

Port 2 Alternate Functions



Memory Map

Figure 3-3. Quick Reference

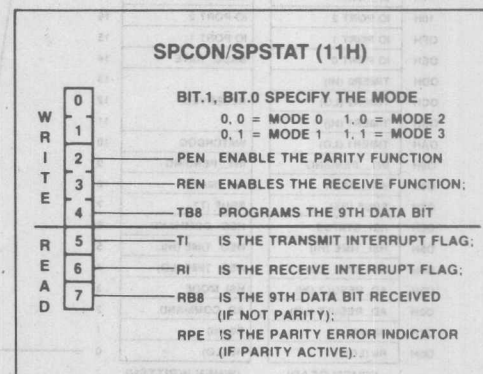
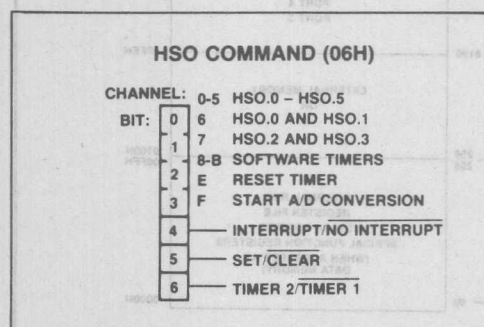
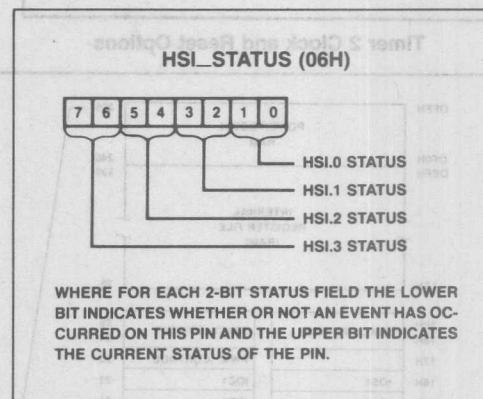
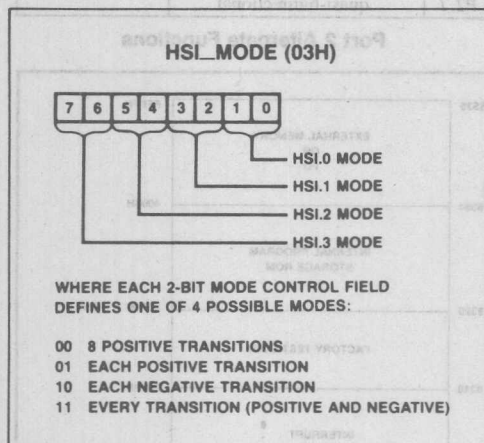
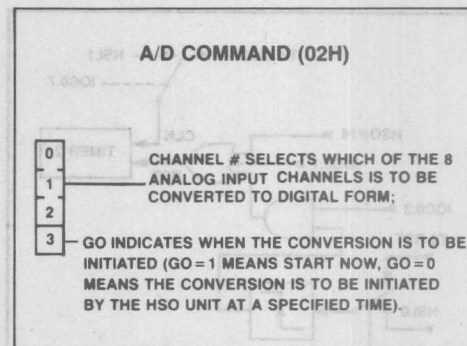
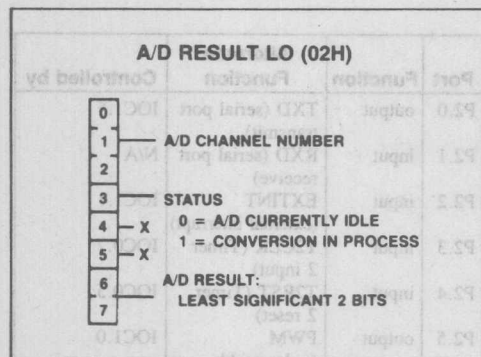


Figure 3-3. (continued)

### IOS0 (15H)

- 0 — HSO.0 CURRENT STATE
- 1 — HSO.1 CURRENT STATE
- 2 — HSO.2 CURRENT STATE
- 3 — HSO.3 CURRENT STATE
- 4 — HSO.4 CURRENT STATE
- 5 — HSO.5 CURRENT STATE
- 6 — CAM OR HOLDING REGISTER IS FULL
- 7 — HSO HOLDING REGISTER IS FULL

### IOC0 (15H)

- 0 — HSI.0 INPUT ENABLE/DISABLE
- 1 — TIMER 2 RESET EACH WRITE
- 2 — HSI.1 INPUT ENABLE/DISABLE
- 3 — TIMER 2 EXTERNAL RESET ENABLE/DISABLE
- 4 — HSI.2 INPUT ENABLE/DISABLE
- 5 — TIMER 2 RESET SOURCE HSI.0/T2 RST
- 6 — HSI.3 INPUT ENABLE/DISABLE
- 7 — TIMER 2 CLOCK SOURCE HSI.1/T2CLK

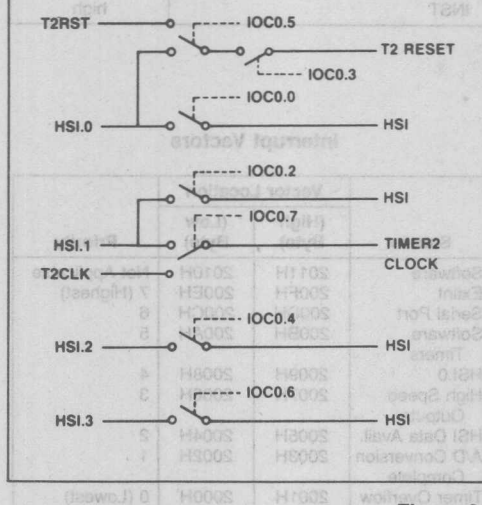
### IOS1 (16H)

- 0 — SOFTWARE TIMER 0 EXPIRED
- 1 — SOFTWARE TIMER 1 EXPIRED
- 2 — SOFTWARE TIMER 2 EXPIRED
- 3 — SOFTWARE TIMER 3 EXPIRED
- 4 — TIMER 2 HAS OVERFLOW
- 5 — TIMER 1 HAS OVERFLOW
- 6 — HSI FIFO IS FULL
- 7 — HSI HOLDING REGISTER DATA AVAILABLE

### IOC1 (16H)

- 0 — SELECT PWM/SELECT P2.5
- 1 — EXTERNAL INTERRUPT ACH7/EXTINT
- 2 — TIMER 1 OVERFLOW INTERRUPT ENABLE/DISABLE
- 3 — TIMER 2 OVERFLOW INTERRUPT ENABLE/DISABLE
- 4 — HSO.4 OUTPUT ENABLE/DISABLE
- 5 — SELECT TXD/SELECT P2.0
- 6 — HSO.5 OUTPUT ENABLE/DISABLE
- 7 — HSI INTERRUPT FIFO FULL/  
HOLDING REGISTER LOADED

### IOC0 (15H)



### Baud Rate Calculations

Using XTAL1:

$$\text{Mode 0: Baud Rate} = \frac{\text{XTAL1 frequency}}{4*(B+1)}, B \neq 0$$

$$\text{Others: Baud Rate} = \frac{\text{XTAL1 frequency}}{64*(B+1)}$$

Using T2CLK:

$$\text{Mode 0: Baud Rate} = \frac{\text{T2CLK frequency}}{B}, B \neq 0$$

$$\text{Others: Baud Rate} = \frac{\text{T2CLK frequency}}{16*B}, B \neq 0$$

Note that B cannot equal 0, except when using XTAL1 in other than mode 0.

Figure 3-3. (continued)

### The PSW Register

|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |          |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----------|
| 15 | 14 | 13 | 12 | 11 | 10 | 09 | 08 | 07 | 06 | 05 | 04 | 03 | 02 | 01 | 00       |
| Z  | N  | V  | VT | C  | —  | I  | ST |    |    |    |    |    |    |    | INT_MASK |

#### WHERE:

Z IS THE ZERO FLAG. IT IS SET WHEN THE RESULT OF AN OPERATION IS ZERO.

N IS THE NEGATIVE FLAG. IT IS SET TO THE ALGEBRAICALLY CORRECT SIGN OF THE RESULT REGARDLESS OF OVERFLOWS.

V IS THE OVERFLOW FLAG. IT IS SET IF AN OVERFLOW OCCURS.

VT IS THE OVERFLOW TRAP FLAG. IT IS SET WHEN THE VT FLAG IS SET AND CLEARED BY JVT, JNVT, OR CLRVT.

C IS THE CARRY FLAG. IT IS SET IF A CARRY WAS GENERATED BY THE PRIOR OPERATION.

I IS THE GLOBAL INTERRUPT ENABLE BIT.

ST IS THE STICKY BIT. IT IS SET DURING A RIGHT SHIFT IF A ONE WAS SHIFTED INTO AND THEN OUT OF THE CARRY FLAG.

INT\_MASK IS THE INTERRUPT MASK REGISTER AND CONTAINS BITS WHICH INDIVIDUALLY ENABLE THE 8 INTERRUPT VECTORS.

#### Reset Status

| Register               | reset value |
|------------------------|-------------|
| Port 1                 | 11111111B   |
| Port 2                 | 110XXXX1B   |
| Port 3                 | 11111111B   |
| Port 4                 | 11111111B   |
| PWM Control            | 00H         |
| Serial Port (Transmit) | undefined   |
| Serial Port (Receive)  | undefined   |
| Baud Rate Register     | undefined   |
| Serial Port Control    | XXXX0XXXB   |
| Serial Port Status     | X00XXXXXB   |
| A/D Command            | undefined   |
| A/D Result             | undefined   |
| Interrupt Pending      | undefined   |
| Interrupt Mask         | 00000000B   |
| Timer 1                | 0000H       |
| Timer 2                | 0000H       |
| Watchdog Timer         | 0000H       |
| HSI Mode               | 11111111B   |
| HSI Status             | undefined   |
| IOS0                   | 00000000B   |
| IOS1                   | 00000000B   |
| IOC0                   | X0X0X0X0B   |
| IOC1                   | X0X0XXX1B   |
| HSI FIFO               | empty       |
| HSO CAM                | empty       |
| HSO lines              | 000000B     |
| PSW                    | 0000H       |
| Stack Pointer          | undefined   |
| Program Counter        | 2080H       |

| Pin  | reset value |
|------|-------------|
| RD   | high        |
| WR   | high        |
| ALE  | low         |
| BHE  | low         |
| INST | high        |

#### Interrupt Vectors

| Source                  | Vector Location |            | Priority       |
|-------------------------|-----------------|------------|----------------|
|                         | (High Byte)     | (Low Byte) |                |
| Software                | 2011H           | 2010H      | Not Applicable |
| Extint                  | 200FH           | 200EH      | 7 (Highest)    |
| Serial Port             | 200DH           | 200CH      | 6              |
| Software                | 200BH           | 200AH      | 5              |
| Timers                  |                 |            |                |
| HSI.0                   | 2009H           | 2008H      | 4              |
| High Speed Outputs      | 2007H           | 2006H      | 3              |
| HSI Data Avail.         | 2005H           | 2004H      | 2              |
| A/D Conversion Complete | 2003H           | 2002H      | 1              |
| Timer Overflow          | 2001H           | 2000H      | 0 (Lowest)     |

Figure 3-3. (continued)



The Following pins are not bonded out in the 48-pin package:

P1.0 through P1.7, P0.0 through P0.3, P2.3, P2.4, P2.6, P2.7 CLKOUT, INST, NMI, TEST, T2CLK(P2.3), T2RST(P2.4).

| Name        | 68-Pin PLCC | 68-Pin PGA | 48-Pin DIP |
|-------------|-------------|------------|------------|
| ACH0/P0.0   | 6           | 4          | —          |
| ACH1/P0.1   | 5           | 5          | —          |
| ACH2/P0.2   | 7           | 3          | —          |
| ACH3/P0.3   | 4           | 6          | —          |
| ACH4/P0.4   | 11          | 67         | 43         |
| ACH5/P0.5   | 10          | 68         | 42         |
| ACH6/P0.6   | 8           | 2          | 40         |
| ACH7/P0.7   | 9           | 1          | 41         |
| ALE         | 62          | 16         | 34         |
| ANGND       | 12          | 66         | 44         |
| BHE         | 41          | 37         | 15         |
| CLKOUT      | 65          | 13         | —          |
| EA          | 2           | 8          | 39         |
| EXTINT/P2.2 | 13          | 63         | 47         |
| HSI.0       | 24          | 54         | 3          |
| HSI.1       | 25          | 53         | 4          |
| HSI.2/HSO.4 | 26          | 52         | 5          |
| HSI.3/HSO.5 | 27          | 51         | 6          |
| HSO.0       | 28          | 50         | 7          |
| HSO.1       | 29          | 49         | 8          |
| HSO.2       | 34          | 44         | 9          |
| HSO.3       | 35          | 43         | 10         |
| HSO.4/HSI.2 | 26          | 52         | 5          |
| HSO.5/HSI.3 | 27          | 51         | 6          |
| INST        | 63          | 15         | —          |
| NMI         | 3           | 7          | —          |
| PWM/P2.5    | 39          | 39         | 13         |
| P0.0/ACH0   | 6           | 4          | —          |
| P0.1/ACH1   | 5           | 5          | —          |
| P0.2/ACH2   | 7           | 3          | —          |
| P0.3/ACH3   | 4           | 6          | —          |
| P0.4/ACH4   | 11          | 67         | 43         |
| P0.5/ACH5   | 10          | 68         | 42         |
| P0.6/ACH6   | 8           | 2          | 40         |
| P0.7/ACH7   | 9           | 1          | 41         |
| P1.0        | 19          | 59         | —          |

## Pin List

Figure 3-3. (continued)

| Name        | 68-Pin PLCC | 68-Pin PGA | 48-Pin DIP |
|-------------|-------------|------------|------------|
| P1.1        | 20          | 58         | —          |
| P1.2        | 21          | 57         | —          |
| P1.3        | 22          | 56         | —          |
| P1.4        | 23          | 55         | —          |
| P1.5        | 30          | 48         | —          |
| P1.6        | 31          | 47         | —          |
| P1.7        | 32          | 46         | —          |
| P2.0/TXD    | 18          | 60         | 2          |
| P2.1/RXD    | 17          | 61         | 1          |
| P2.2/EXTINT | 15          | 63         | 47         |
| P2.3/T2CLK  | 44          | 34         | —          |
| P2.4/T2RST  | 42          | 36         | —          |
| P2.5/PWM    | 39          | 39         | 13         |
| P2.6        | 33          | 45         | —          |
| P2.7        | 38          | 40         | —          |
| P3.0/AD0    | 60          | 18         | 32         |
| P3.1/AD1    | 59          | 19         | 31         |
| P3.2/AD2    | 58          | 20         | 30         |
| P3.3/AD3    | 57          | 21         | 29         |
| P3.4/AD4    | 56          | 22         | 28         |
| P3.5/AD5    | 55          | 23         | 27         |
| P3.6/AD6    | 54          | 24         | 26         |
| P3.7/AD7    | 53          | 25         | 25         |
| P4.0/AD8    | 52          | 26         | 24         |
| P4.1/AD9    | 51          | 27         | 23         |
| P4.2/AD10   | 50          | 28         | 22         |
| P4.3/AD11   | 49          | 29         | 21         |
| P4.4/AD12   | 48          | 30         | 20         |
| P4.5/AD13   | 47          | 31         | 19         |
| P4.6/AD14   | 46          | 32         | 18         |
| P4.7/AD15   | 45          | 33         | 17         |
| RD          | 61          | 17         | 33         |
| READY       | 43          | 35         | 16         |
| RESET       | 16          | 62         | 48         |
| RXD/P2.1    | 17          | 61         | 1          |
| TEST        | 64          | 14         | —          |
| TXD/P2.0    | 18          | 60         | 2          |
| T2CLK/P2.3  | 44          | 34         | —          |
| T2RST/P2.4  | 42          | 36         | —          |
| VBB         | 37          | 41         | 12         |
| VCC         | 1           | 9          | 38         |
| VPD         | 14          | 64         | 46         |
| VREF        | 13          | 65         | 45         |
| VSS         | 68          | 10         | 11         |
| VSS         | 36          | 42         | 37         |
| WR          | 40          | 38         | 14         |
| XTAL1       | 67          | 11         | 36         |
| XTAL2       | 66          | 12         | 35         |

### 3.5. SOFTWARE STANDARDS AND CONVENTIONS

For a software project of any size it is a good idea to modularize the program and to establish standards which control the communication between these modules. The nature of these standards will vary with the needs of the final application. A common component of all of these standards, however, must be the mechanism for passing parameters to procedures and returning results from procedures. In the absence of some overriding consideration which prevents their use, it is suggested that the user conform to the conventions adopted by the PLM-96 programming language for procedure linkage. It is a very usable standard for both the assembly language and PLM-96 environment and it offers compatibility between these environments. Another advantage is that it allows the user access to the same floating point arithmetics library that PLM-96 uses to operate on REAL variables.

#### 3.5.1. Register Utilization

The MCS-96 architecture provides a 256 byte register file. Some of these registers are used to control register-mapped I/O devices and for other special functions such as the ZERO register and the stack pointer. The remaining bytes in the register file, some 230 of them, are available for allocation by the programmer. If these registers are to be used effectively some overall strategy for their allocation must be adopted. PLM-96 adopts the simple and effective strategy of allocating the eight bytes between addresses 1CH and 23H as temporary storage. The starting address of this region is called PLMREG. The remaining area in the register file is treated as a segment of memory which is allocated as required.

#### 3.5.2. Addressing 32-bit Operands

These operands are formed from two adjacent 16-bit words in memory. The least significant word of the double word is always in lower address, even when the data is in the stack (which means that the most significant word must be pushed into the stack first). A double word is addressed by the address of its least significant byte. Note that the hardware supports some operations on double words (e.g. normalize and divide). For these operations the double word must be in the internal register file and must have an address which is evenly divisible by four.

#### 3.5.3. Subroutine Linkage

Parameters are passed to subroutines in the stack. Parameters are pushed into the stack in the order that they are encountered in the scanning of the source text. Eight-bit parameters (BYTES or SHORT-INTEGERS) are pushed into the stack with the high order byte undefined. Thirty-two bit parameters (LONG-INTEGERS, DOUBLE-WORDS, and REALS) are pushed into the stack as two 16 bit values; the most significant half of the parameter is pushed into the stack first.

As an example, consider the following PLM-96 procedure:

```
example_procedure: PROCEDURE (param1,param2,param3);
    DECLARE param1 BYTE,
              param2 DWORD,
              param3 WORD;
```

When this procedure is entered at run time the stack will contain the parameters in the following order:

|              |                |
|--------------|----------------|
| ????? :      | param1         |
| high word of | param2         |
| low word of  | param2         |
|              | param3         |
|              | return address |

← Stack\_pointer

Figure 3-4. Stack Image

If a procedure returns a value to the calling code (as opposed to modifying more global variables) then the result is returned in the variable PLMREG. PLMREG is viewed as either an 8, 16 or 32 bit variable depending on the type of the procedure.

The standard calling convention adopted by PLM-96 has several key features:

- Procedures can always assume that the eight bytes of register file memory starting at PLMREG can be used as temporaries within the body of the procedure.
- Code which calls a procedure must assume that the eight bytes of register file memory starting at PLMREG are modified by the procedure.
- The Program Status Word (PSW-see section 3.3) is not saved and restored by procedures so the calling code must assumed that the condition flags (Z,N,V,VT,C, and ST) are modified by the procedure.
- Function results from procedures are always returned in the variable PLMREG.

PLM-96 allows the definition of INTERRUPT procedures which are executed when a predefined interrupt occurs. These procedures do not conform to the rules of a normal procedure. Parameters cannot be passed to these procedures and they cannot return results. Since they can execute essentially at any time (hence the term interrupt), these procedures must save the PSW and PLMREG when they are entered and restore these values before they exit.

### 3.6. USING THE INTERRUPT SYSTEM

Processing interrupts is an integral part of almost any control application. The 8096 allows the program to manage interrupt servicing in an efficient and flexible manner. Software running in the 8096 exerts control over the interrupt hardware at several levels.

### 3.6.1. Global Lockout

The processing of interrupts can be enabled or disabled by setting or clearing the I bit in the PSW. This is accomplished by the EI (Enable Interrupts) and DI (Disable Interrupts) instructions. Note that the I bit only controls the actual servicing of interrupts; interrupts that occur during periods of lockout will be held in the pending register and serviced on a prioritized basis when the lockout period ends.

### 3.6.2. Pending Interrupt Register

When the hardware detects one of the eight interrupts it sets the corresponding bit in the pending interrupt register (INT\_PENDING-register 09H). This register, which has the same bit layout as the interrupt mask register (see next section), can be read or modified as a byte register. This register can be read to determine which of the interrupts are pending at any given time or modified to either clear pending interrupts or generate interrupts under software control. Any software which modifies the INT\_PENDING register should ensure that the entire operation is indivisible. The easiest way of doing this is to use the logical instructions in the two or three operand format, as examples:

```

ANDB INT_PENDING, #11111101B
ORB   ; Clears the A/D interrupt
INT_PENDING, #00000010B
      ; Sets the A/D interrupt

```

If the required modification to INT\_PENDING cannot be accomplished with one instruction then a critical region should be established and the INT\_PENDING register modified from within this region (see section 3.6.5).

### 3.6.3. Interrupt Mask Register

Individual interrupts can be enabled or disabled by setting or clearing bits in the interrupt mask register (INT\_MASK-register 08H). The format of this register is shown in figure 3-5.

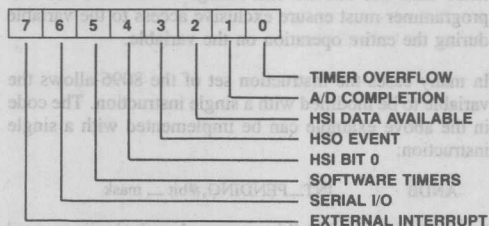


Figure 3-5. Interrupt Mask Register

The INT\_MASK register can be read or written as a byte register. A one in any bit position will enable the corresponding interrupt source and a zero will disable the source. The individual masks act like the global lockout in that they only control the servicing of the interrupt; the hardware will save any interrupts that occur in the pending register even if the interrupt mask bit is cleared. The INT\_MASK register also can be accessed as the lower

eight bits of the PSW so the PUSHF and POPF instructions save and restore the INT\_MASK register as well as the global interrupt lockout and the arithmetic flags.

### 3.6.4. Interrupt Vectors

The 8096 has eight sources of hardware interrupt, each with its own priority and interrupt vector location. Table 3-5 shows the interrupt sources, their priority, and their vector locations. See section 2.5 for a discussion of the various interrupt sources.

Table 3-5. Interrupt Vector Information

| Source             | Priority  | Vector |
|--------------------|-----------|--------|
| Timer Overflow     | 0-Lowest  | 2000H  |
| A/D Completion     | 1         | 2002H  |
| HSI Data Available | 2         | 2004H  |
| HSO Execution      | 3         | 2006H  |
| HSI.O              | 4         | 2008H  |
| Software timers    | 5         | 200AH  |
| Serial I/O         | 6         | 200CH  |
| External Interrupt | 7-Highest | 200EH  |

The programmer must initialize the interrupt vector table with the starting addresses of the appropriate interrupt service routine. It would be a good idea to vector any interrupts that are not used in the system to an error handling routine.

The priorities given in the table give the hardware enforced priorities for these interrupts. This priority controls the order in which pending interrupts are passed to the software via interrupt-calls. The software can implement its own priority structure by controlling the mask register (INT\_MASK-register 08H). To see how this is done consider the case of a serial I/O service routine which must run at a priority level which is lower than the HSI data available interrupt but higher than any other source. The "preamble" and exit code for this interrupt service routine would look like this:

```

serial _ io _ isr:
    PUSHF          ; Save the PSW
                   ; (Includes INT_MASK)
    LDB INT_MASK, #00000100B
    EI             ; Enable interrupts again

    ; Service the interrupt

    POPF
    RET            ; Restore the PSW

```

Note that location 200CH in the interrupt vector table would have to be loaded with the value of the label serial \_ io \_ isr and the interrupt be enabled for this routine to execute.

which makes this (or any other) 8096 interrupt service routine execute properly:

a). After the hardware decides to process an interrupt it generates and executes a special interrupt-call instruction which pushes the current program counter onto the stack and then loads the program counter with the contents of the vector table entry corresponding to the interrupt. The hardware will not allow another interrupt to be serviced immediately following the interrupt-call. This guarantees that once the interrupt-call starts the first instruction of the interrupt service routine will execute.

b). The PUSHF instruction, which is now guaranteed to execute, saves the PSW in the stack and then clears the PSW. The PSW contains, in addition to the arithmetic flags, the INT\_MASK register and the global enable flag (I). The hardware will not allow an interrupt following a PUSHF instruction and by the time the LD instruction starts all of the interrupt enable flags will be cleared. Now there is guaranteed execution of the LD INT\_MASK instruction.

c). The LD INT\_MASK instruction enables those interrupts that the programmer chooses to allow to interrupt the serial I/O interrupt service routine. In this example only the HSI data available interrupt will be allowed to do this but any interrupt or combination of interrupts could be enabled at this point, even the serial interrupt. It is the loading of the INT\_MASK register which allows the software to establish its own priorities for interrupt servicing independently from those that the hardware enforces.

d). The EI instruction reenables the processing of interrupts.

e). The actual interrupt service routine executes within the priority structure established by the software.

f). At the end of the service routine the POPF instruction restores the PSW to its state when the interrupt-call occurred. The hardware will not allow interrupts to be processed following a POPF instruction so the execution of the last instruction (RET) is guaranteed before further interrupts can occur. The reason that this RET instruction must be protected in this fashion is that it is quite likely that the POPF instruction will reenables an interrupt which is already pending. If this interrupt were serviced before the RET instruction, then the return address to the code that was executing when the original interrupt occurred would be left on the stack. While this does not present a problem to the program flow, it could result in a stack overflow if interrupts are occurring at a high frequency. The POPF instruction also pops the INT\_MASK register (part of the PSW), so any changes made to this register during a routine which ends with a POPF will be lost.

service routine does not include any code for saving or restoring registers. This is because it has been assumed that the interrupt service routine has been allocated its own private set of registers from the on-board register file. The availability of some 230 bytes of register storage makes this quite practical.

### 3.6.5. Critical Regions

Interrupt service routines must share some data with other routines. Whenever the programmer is coding those sections of code which access these shared pieces of data, great care must be taken to ensure that the integrity of the data is maintained. Consider clearing a bit in the interrupt pending register as part of a non-interrupt routine:

```
LDB    AL,INT_PENDING
ANDB   AL,#bit_mask
STB    AL,INT_PENDING
```

This code works if no other routines are operating concurrently, but will cause occasional but serious problems if used in a concurrent environment. (All programs which make use of interrupts must be considered to be part of a concurrent environment.) To demonstrate this problem, assume that the INT\_PENDING register contains 00001111B and bit 3 (HSO event interrupt pending) is to be reset. The code does work for this data pattern but what happens if an HSI interrupt occurs somewhere between the LDB and the STB instructions? Before the LDB instruction INT\_PENDING contains 00001111B and after the LDB instruction so does AL. If the HSI interrupt service routine executes at this point then INT\_PENDING will change to 00001011B. The ANDB changes AL to 00000111B and the STB changes INT\_PENDING to 00000111B. It should be 00000011B. This code sequence has managed to generate a false HSI interrupt! The same basic process can generate an amazing assortment of problems and headaches. These problems can be avoided by assuring mutual exclusion which basically means that if more than one routine can change a variable, then the programmer must ensure exclusive access to the variable during the entire operation on the variable.

In many cases the instruction set of the 8096 allows the variable to be modified with a single instruction. The code in the above example can be implemented with a single instruction:

```
ANDB   INT_PENDING,#bit_mask
```

Instructions are indivisible so mutual exclusion is ensured in this case. For more complex situations, such a simple solution is not available and the programmer must create what is termed a critical region in which it is safe to modify the variable. One way to do this is to simply disable interrupts with a DI instruction, perform the modification, and then re-enable interrupts with an EI instruction. The problem with this approach is that it leaves the interrupts enabled even if they were not enabled at the start. A better solution is to enter the critical region with a PUSHF instruction which saves the PSW and also clears



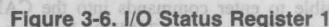
The on-board I/O devices are, for the most part, simple to program. There are some areas of potential confusion which need to be addressed:

Some of the on-board I/O ports can be used as both input and output pins (e.g. Port 1). When the processor writes to the pins of these ports it actually writes into a register which in turn drives the port pin. When the processor reads these ports, it senses the status of the pin directly. If a port pin is to be used as an input then the software should write a one to that pin, this will cause the low-impedance pull-down device to turn off and leave the pin pulled up with a relatively high impedance pull-up device which can be easily driven down by the device driving the input. If some pins of a port are to be used as inputs and some are to be used as outputs the programmer should be careful when writing to the port. Consider using P1.0 as an input and then trying to toggle P1.1 as an output:

The first instruction will work as expected but two problems can occur when the second instruction executes. The first is that even though P1.1 is being driven high by the 8096 it is possible that it is being held low externally. This typically happens when the port pin is used to drive the base of an NPN transistor which in turn drives whatever there is in the outside world which needs to be toggled. The base of the transistor will clamp the port pin to the transistor's  $V_{be}$  above ground, typically 0.7 volts. The 8096 will input this value as a zero even if a one has been written to the port pin. When this happens the XORB instruction will always write a one to the port pin and it will not toggle. The second problem, which is related to the first one, is that if P1.0 happens to be driven to a zero when Port 1 is read by the XORB instruction then the XORB will write a zero to P1.0 and it will no longer be useable as an input. The first problem can best be solved by the external driver design. A series resistor between the port pin and the base of the transistor often works. The second problem can be solved in the software fairly easily:

A software solution to both problems is to keep a byte in RAM as an image of the data to be output to the port; any time the software wants to modify the data on the port it can then modify the image byte and then copy it to the port.

This status register contains a collection of status flags which relate to the timer and high speed I/O functions (see section 2.12.5). It can be accessed as register 16H in the on-board register file. The layout of this register is shown in figure 3-6.



LDB ALJOS1

JB IOS1.3.somewhere \_ else

ORB      IOS1 \_ image,IOS1

leaving `IOS1_image` containing all the flags that were set before plus all the new flags that were read and cleared from `IOS1_image`. Any other routine which needs to sample the flags can safely check `IOS1_image`. Note that if these routines need to clear the flags that they have acted on then the modification of `IOS1_image` must be done from inside a critical region (see section 3.6.5).

Commands are sent to the HSO unit via a byte and then a word write operation:

3-19

The command is actually accepted when the HSO — TIME register is written. It is important to ensure that this code piece is not interrupted by any interrupt service routine which might also send a command to the HSO unit. If this happens the HSO will know when to do it but not know what to do when it's time to do it. In many systems this becomes a null problem because HSO commands are only issued from one place in the code. If this is not the case then a critical region must be established and the two instructions executed from within this region (see section 3.6.5).

Commands in the holding register will not execute even if their time tag is reached. Commands must be in the CAM for this to occur. Commands in the holding register can also be overwritten. Since it can take up to 8 state times for a command to move from the holding register to the CAM, 8 states must be allowed between successive writes to the CAM. Flags are available in IOS0 which indicate the holding register is empty (IOS0.7) or that both the holding register is empty and the CAM is not full (IOS0.6). The programmer should carefully decide which of these two flags is the best to use for each application.

It is possible to enter commands into the CAM which never execute. This occurs if TIMER2 has been set up as a variable modulo counter and a command is entered with a time tag referenced to TIMER2 which has a value that TIMER2 never reaches. The inaccessible command will never execute and continue to take up room in the CAM until either the system is reset or the program allows TIMER2 increment up to the value stored in the time tag. Note that commands cannot be flushed from the CAM without being executed but that they can be cancelled. This is accomplished by setting the opposite command in the CAM to execute at the same time tag as the command to be cancelled. Since internal events are not synchronized to Timer 1, it is not possible to cancel them. If, as an example, a command has been issued to set HSO.1 when TIMER1 = 1234 then entering a second command which clears HSO.1 when TIMER1 = 1234 will result in a no-operation on HSO.1. Both commands will remain in the CAM until TIMER1 = 1234.

### 3.7.4. High Speed I/O Interrupts

The HSO unit can generate two types of interrupts. The HSO execution interrupt (vector = (2006H)) is generated (if enabled) for HSO commands which operate on one of the six HSO pins. The other HSO interrupt is the Software Timer interrupt (vector = (200AH)) which is generated (if enabled) for any other HSO command (e.g. triggering the A/D, resetting Timer2 or generating a software time delay).

There are also two interrupts associated with the HSI unit. The HSI data available interrupt (vector = (2004H)) is generated if there is data in the HSI FIFO that the program should read. The other HSI related interrupt is the HSI.0 interrupt which occurs whenever High Speed Input pin 0 makes a zero-to-one transition. This interrupt will become pending in the INT\_PENDING register even if the HSI unit is programmed to ignore changes on HSI.0 or look for a one-to-zero transition.

### 3.7.5. Accessing Register Mapped I/O

The on-board I/O devices such as the serial port or the A/D converter are controlled as register mapped I/O. This allows convenient and efficient I/O processing. The implementation of the current members of the MCS-96 family place some restrictions on how these registers can be accessed. While these restrictions are not severe, the programmer must be aware of them. A complete listing of these registers is shown in figure 2-7 and 2-8. The restrictions are as follows:

- TIMER1, TIMER2 and HSI\_TIME are *word read only*. They cannot be read as bytes or written to in any format.
- HSO — TIME is *word write only*. It cannot be written to as individual bytes or read in any format.
- R0 (the ZERO register) is byte or word read or write but writing to it will not change its value.
- All of the other I/O registers can be accessed only as bytes. This applies even to the AD\_RESULT which is logically a word operand.
- Neither the source nor the destination addresses of the Multiply and Divide instructions can be a writable special function register.
- These registers may not be used as base or index registers for indexed or indirect instructions.

3.7.3. Sending Commands to the HSO Unit  
Commands are sent to the HSO unit via a byte and then a word write operation.

```
LDI  R20, #0
MOV  R21, R20
MOV  R22, R20
MOV  R23, R20
MOV  R24, R20
MOV  R25, R20
MOV  R26, R20
MOV  R27, R20
MOV  R28, R20
MOV  R29, R20
MOV  R30, R20
MOV  R31, R20
MOV  R32, R20
MOV  R33, R20
MOV  R34, R20
MOV  R35, R20
MOV  R36, R20
MOV  R37, R20
MOV  R38, R20
MOV  R39, R20
MOV  R40, R20
MOV  R41, R20
MOV  R42, R20
MOV  R43, R20
MOV  R44, R20
MOV  R45, R20
MOV  R46, R20
MOV  R47, R20
MOV  R48, R20
MOV  R49, R20
MOV  R50, R20
MOV  R51, R20
MOV  R52, R20
MOV  R53, R20
MOV  R54, R20
MOV  R55, R20
MOV  R56, R20
MOV  R57, R20
MOV  R58, R20
MOV  R59, R20
MOV  R60, R20
MOV  R61, R20
MOV  R62, R20
MOV  R63, R20
MOV  R64, R20
MOV  R65, R20
MOV  R66, R20
MOV  R67, R20
MOV  R68, R20
MOV  R69, R20
MOV  R70, R20
MOV  R71, R20
MOV  R72, R20
MOV  R73, R20
MOV  R74, R20
MOV  R75, R20
MOV  R76, R20
MOV  R77, R20
MOV  R78, R20
MOV  R79, R20
MOV  R80, R20
MOV  R81, R20
MOV  R82, R20
MOV  R83, R20
MOV  R84, R20
MOV  R85, R20
MOV  R86, R20
MOV  R87, R20
MOV  R88, R20
MOV  R89, R20
MOV  R90, R20
MOV  R91, R20
MOV  R92, R20
MOV  R93, R20
MOV  R94, R20
MOV  R95, R20
MOV  R96, R20
MOV  R97, R20
MOV  R98, R20
MOV  R99, R20
MOV  R100, R20
MOV  R101, R20
MOV  R102, R20
MOV  R103, R20
MOV  R104, R20
MOV  R105, R20
MOV  R106, R20
MOV  R107, R20
MOV  R108, R20
MOV  R109, R20
MOV  R110, R20
MOV  R111, R20
MOV  R112, R20
MOV  R113, R20
MOV  R114, R20
MOV  R115, R20
MOV  R116, R20
MOV  R117, R20
MOV  R118, R20
MOV  R119, R20
MOV  R120, R20
MOV  R121, R20
MOV  R122, R20
MOV  R123, R20
MOV  R124, R20
MOV  R125, R20
MOV  R126, R20
MOV  R127, R20
MOV  R128, R20
MOV  R129, R20
MOV  R130, R20
MOV  R131, R20
MOV  R132, R20
MOV  R133, R20
MOV  R134, R20
MOV  R135, R20
MOV  R136, R20
MOV  R137, R20
MOV  R138, R20
MOV  R139, R20
MOV  R140, R20
MOV  R141, R20
MOV  R142, R20
MOV  R143, R20
MOV  R144, R20
MOV  R145, R20
MOV  R146, R20
MOV  R147, R20
MOV  R148, R20
MOV  R149, R20
MOV  R150, R20
MOV  R151, R20
MOV  R152, R20
MOV  R153, R20
MOV  R154, R20
MOV  R155, R20
MOV  R156, R20
MOV  R157, R20
MOV  R158, R20
MOV  R159, R20
MOV  R160, R20
MOV  R161, R20
MOV  R162, R20
MOV  R163, R20
MOV  R164, R20
MOV  R165, R20
MOV  R166, R20
MOV  R167, R20
MOV  R168, R20
MOV  R169, R20
MOV  R170, R20
MOV  R171, R20
MOV  R172, R20
MOV  R173, R20
MOV  R174, R20
MOV  R175, R20
MOV  R176, R20
MOV  R177, R20
MOV  R178, R20
MOV  R179, R20
MOV  R180, R20
MOV  R181, R20
MOV  R182, R20
MOV  R183, R20
MOV  R184, R20
MOV  R185, R20
MOV  R186, R20
MOV  R187, R20
MOV  R188, R20
MOV  R189, R20
MOV  R190, R20
MOV  R191, R20
MOV  R192, R20
MOV  R193, R20
MOV  R194, R20
MOV  R195, R20
MOV  R196, R20
MOV  R197, R20
MOV  R198, R20
MOV  R199, R20
MOV  R200, R20
MOV  R201, R20
MOV  R202, R20
MOV  R203, R20
MOV  R204, R20
MOV  R205, R20
MOV  R206, R20
MOV  R207, R20
MOV  R208, R20
MOV  R209, R20
MOV  R210, R20
MOV  R211, R20
MOV  R212, R20
MOV  R213, R20
MOV  R214, R20
MOV  R215, R20
MOV  R216, R20
MOV  R217, R20
MOV  R218, R20
MOV  R219, R20
MOV  R220, R20
MOV  R221, R20
MOV  R222, R20
MOV  R223, R20
MOV  R224, R20
MOV  R225, R20
MOV  R226, R20
MOV  R227, R20
MOV  R228, R20
MOV  R229, R20
MOV  R230, R20
MOV  R231, R20
MOV  R232, R20
MOV  R233, R20
MOV  R234, R20
MOV  R235, R20
MOV  R236, R20
MOV  R237, R20
MOV  R238, R20
MOV  R239, R20
MOV  R240, R20
MOV  R241, R20
MOV  R242, R20
MOV  R243, R20
MOV  R244, R20
MOV  R245, R20
MOV  R246, R20
MOV  R247, R20
MOV  R248, R20
MOV  R249, R20
MOV  R250, R20
MOV  R251, R20
MOV  R252, R20
MOV  R253, R20
MOV  R254, R20
MOV  R255, R20
MOV  R256, R20
MOV  R257, R20
MOV  R258, R20
MOV  R259, R20
MOV  R260, R20
MOV  R261, R20
MOV  R262, R20
MOV  R263, R20
MOV  R264, R20
MOV  R265, R20
MOV  R266, R20
MOV  R267, R20
MOV  R268, R20
MOV  R269, R20
MOV  R270, R20
MOV  R271, R20
MOV  R272, R20
MOV  R273, R20
MOV  R274, R20
MOV  R275, R20
MOV  R276, R20
MOV  R277, R20
MOV  R278, R20
MOV  R279, R20
MOV  R280, R20
MOV  R281, R20
MOV  R282, R20
MOV  R283, R20
MOV  R284, R20
MOV  R285, R20
MOV  R286, R20
MOV  R287, R20
MOV  R288, R20
MOV  R289, R20
MOV  R290, R20
MOV  R291, R20
MOV  R292, R20
MOV  R293, R20
MOV  R294, R20
MOV  R295, R20
MOV  R296, R20
MOV  R297, R20
MOV  R298, R20
MOV  R299, R20
MOV  R300, R20
MOV  R301, R20
MOV  R302, R20
MOV  R303, R20
MOV  R304, R20
MOV  R305, R20
MOV  R306, R20
MOV  R307, R20
MOV  R308, R20
MOV  R309, R20
MOV  R310, R20
MOV  R311, R20
MOV  R312, R20
MOV  R313, R20
MOV  R314, R20
MOV  R315, R20
MOV  R316, R20
MOV  R317, R20
MOV  R318, R20
MOV  R319, R20
MOV  R320, R20
MOV  R321, R20
MOV  R322, R20
MOV  R323, R20
MOV  R324, R20
MOV  R325, R20
MOV  R326, R20
MOV  R327, R20
MOV  R328, R20
MOV  R329, R20
MOV  R330, R20
MOV  R331, R20
MOV  R332, R20
MOV  R333, R20
MOV  R334, R20
MOV  R335, R20
MOV  R336, R20
MOV  R337, R20
MOV  R338, R20
MOV  R339, R20
MOV  R340, R20
MOV  R341, R20
MOV  R342, R20
MOV  R343, R20
MOV  R344, R20
MOV  R345, R20
MOV  R346, R20
MOV  R347, R20
MOV  R348, R20
MOV  R349, R20
MOV  R350, R20
MOV  R351, R20
MOV  R352, R20
MOV  R353, R20
MOV  R354, R20
MOV  R355, R20
MOV  R356, R20
MOV  R357, R20
MOV  R358, R20
MOV  R359, R20
MOV  R360, R20
MOV  R361, R20
MOV  R362, R20
MOV  R363, R20
MOV  R364, R20
MOV  R365, R20
MOV  R366, R20
MOV  R367, R20
MOV  R368, R20
MOV  R369, R20
MOV  R370, R20
MOV  R371, R20
MOV  R372, R20
MOV  R373, R20
MOV  R374, R20
MOV  R375, R20
MOV  R376, R20
MOV  R377, R20
MOV  R378, R20
MOV  R379, R20
MOV  R380, R20
MOV  R381, R20
MOV  R382, R20
MOV  R383, R20
MOV  R384, R20
MOV  R385, R20
MOV  R386, R20
MOV  R387, R20
MOV  R388, R20
MOV  R389, R20
MOV  R390, R20
MOV  R391, R20
MOV  R392, R20
MOV  R393, R20
MOV  R394, R20
MOV  R395, R20
MOV  R396, R20
MOV  R397, R20
MOV  R398, R20
MOV  R399, R20
MOV  R400, R20
MOV  R401, R20
MOV  R402, R20
MOV  R403, R20
MOV  R404, R20
MOV  R405, R20
MOV  R406, R20
MOV  R407, R20
MOV  R408, R20
MOV  R409, R20
MOV  R410, R20
MOV  R411, R20
MOV  R412, R20
MOV  R413, R20
MOV  R414, R20
MOV  R415, R20
MOV  R416, R20
MOV  R417, R20
MOV  R418, R20
MOV  R419, R20
MOV  R420, R20
MOV  R421, R20
MOV  R422, R20
MOV  R423, R20
MOV  R424, R20
MOV  R425, R20
MOV  R426, R20
MOV  R427, R20
MOV  R428, R20
MOV  R429, R20
MOV  R430, R20
MOV  R431, R20
MOV  R432, R20
MOV  R433, R20
MOV  R434, R20
MOV  R435, R20
MOV  R436, R20
MOV  R437, R20
MOV  R438, R20
MOV  R439, R20
MOV  R440, R20
MOV  R441, R20
MOV  R442, R20
MOV  R443, R20
MOV  R444, R20
MOV  R445, R20
MOV  R446, R20
MOV  R447, R20
MOV  R448, R20
MOV  R449, R20
MOV  R450, R20
MOV  R451, R20
MOV  R452, R20
MOV  R453, R20
MOV  R454, R20
MOV  R455, R20
MOV  R456, R20
MOV  R457, R20
MOV  R458, R20
MOV  R459, R20
MOV  R460, R20
MOV  R461, R20
MOV  R462, R20
MOV  R463, R20
MOV  R464, R20
MOV  R465, R20
MOV  R466, R20
MOV  R467, R20
MOV  R468, R20
MOV  R469, R20
MOV  R470, R20
MOV  R471, R20
MOV  R472, R20
MOV  R473, R20
MOV  R474, R20
MOV  R475, R20
MOV  R476, R20
MOV  R477, R20
MOV  R478, R20
MOV  R479, R20
MOV  R480, R20
MOV  R481, R20
MOV  R482, R20
MOV  R483, R20
MOV  R484, R20
MOV  R485, R20
MOV  R486, R20
MOV  R487, R20
MOV  R488, R20
MOV  R489, R20
MOV  R490, R20
MOV  R491, R20
MOV  R492, R20
MOV  R493, R20
MOV  R494, R20
MOV  R495, R20
MOV  R496, R20
MOV  R497, R20
MOV  R498, R20
MOV  R499, R20
MOV  R500, R20
MOV  R501, R20
MOV  R502, R20
MOV  R503, R20
MOV  R504, R20
MOV  R505, R20
MOV  R506, R20
MOV  R507, R20
MOV  R508, R20
MOV  R509, R20
MOV  R510, R20
MOV  R511, R20
MOV  R512, R20
MOV  R513, R20
MOV  R514, R20
MOV  R515, R20
MOV  R516, R20
MOV  R517, R20
MOV  R518, R20
MOV  R519, R20
MOV  R520, R20
MOV  R521, R20
MOV  R522, R20
MOV  R523, R20
MOV  R524, R20
MOV  R525, R20
MOV  R526, R20
MOV  R527, R20
MOV  R528, R20
MOV  R529, R20
MOV  R530, R20
MOV  R531, R20
MOV  R532, R20
MOV  R533, R20
MOV  R534, R20
MOV  R535, R20
MOV  R536, R20
MOV  R537, R20
MOV  R538, R20
MOV  R539, R20
MOV  R540, R20
MOV  R541, R20
MOV  R542, R20
MOV  R543, R20
MOV  R544, R20
MOV  R545, R20
MOV  R546, R20
MOV  R547, R20
MOV  R548, R20
MOV  R549, R20
MOV  R550, R20
MOV  R551, R20
MOV  R552, R20
MOV  R553, R20
MOV  R554, R20
MOV  R555, R20
MOV  R556, R20
MOV  R557, R20
MOV  R558, R20
MOV  R559, R20
MOV  R560, R20
MOV  R561, R20
MOV  R562, R20
MOV  R563, R20
MOV  R564, R20
MOV  R565, R20
MOV  R566, R20
MOV  R567, R20
MOV  R568, R20
MOV  R569, R20
MOV  R570, R20
MOV  R571, R20
MOV  R572, R20
MOV  R573, R20
MOV  R574, R20
MOV  R575, R20
MOV  R576, R20
MOV  R577, R20
MOV  R578, R20
MOV  R579, R20
MOV  R580, R20
MOV  R581, R20
MOV  R582, R20
MOV  R583, R20
MOV  R584, R20
MOV  R585, R20
MOV  R586, R20
MOV  R587, R20
MOV  R588, R20
MOV  R589, R20
MOV  R590, R20
MOV  R591, R20
MOV  R592, R20
MOV  R593, R20
MOV  R594, R20
MOV  R595, R20
MOV  R596, R20
MOV  R597, R20
MOV  R598, R20
MOV  R599, R20
MOV  R600, R20
MOV  R601, R20
MOV  R602, R20
MOV  R603, R20
MOV  R604, R20
MOV  R605, R20
MOV  R606, R20
MOV  R607, R20
MOV  R608, R20
MOV  R609, R20
MOV  R610, R20
MOV  R611, R20
MOV  R612, R20
MOV  R613, R20
MOV  R614, R20
MOV  R615, R20
MOV  R616, R20
MOV  R617, R20
MOV  R618, R20
MOV  R619, R20
MOV  R620, R20
MOV  R621, R20
MOV  R622, R20
MOV  R623, R20
MOV  R624, R20
MOV  R625, R20
MOV  R626, R20
MOV  R627, R20
MOV  R628, R20
MOV  R629, R20
MOV  R630, R20
MOV  R631, R20
MOV  R632, R20
MOV  R633, R20
MOV  R634, R20
MOV  R635, R20
MOV  R636, R20
MOV  R637, R20
MOV  R638, R20
MOV  R639, R20
MOV  R640, R20
MOV  R641, R20
MOV  R642, R20
MOV  R643, R20
MOV  R644, R20
MOV  R645, R20
MOV  R646, R20
MOV  R647, R20
MOV  R648, R20
MOV  R649, R20
MOV  R650, R20
MOV  R651, R20
MOV  R652, R20
MOV  R653, R20
MOV  R654, R20
MOV  R655, R20
MOV  R656, R20
MOV  R657, R20
MOV  R658, R20
MOV  R659, R20
MOV  R660, R20
MOV  R661, R20
MOV  R662, R20
MOV  R663, R20
MOV  R664, R20
MOV  R665, R20
MOV  R666, R20
MOV  R667, R20
MOV  R668, R20
MOV  R669, R20
MOV  R670, R20
MOV  R671, R20
MOV  R672, R20
MOV  R673, R20
MOV  R674, R20
MOV  R675, R20
MOV  R676, R20
MOV  R677, R20
MOV  R678, R20
MOV  R679, R20
MOV  R680, R20
MOV  R681, R20
MOV  R682, R20
MOV  R683, R20
MOV  R684, R20
MOV  R685, R20
MOV  R686, R20
MOV  R687, R20
MOV  R688, R20
MOV  R689, R20
MOV  R690, R20
MOV  R691, R20
MOV  R692, R20
MOV  R693, R20
MOV  R694, R20
MOV  R695, R20
MOV  R696, R20
MOV  R697, R20
MOV  R698, R20
MOV  R699, R20
MOV  R700, R20
MOV  R701, R20
MOV  R702, R20
MOV  R703, R20
MOV  R704, R20
MOV  R705, R20
MOV  R706, R20
MOV  R707, R20
MOV  R708, R20
MOV  R709, R20
MOV  R710, R20
MOV  R711, R20
MOV  R712, R20
MOV  R713, R20
MOV  R714, R20
MOV  R715, R20
MOV  R716, R20
MOV  R717, R20
MOV  R718, R20
MOV  R719, R20
MOV  R720, R20
MOV  R721, R20
MOV  R722, R20
MOV  R723, R20
MOV  R724, R20
MOV  R725, R20
MOV  R726, R20
MOV  R727, R20
MOV  R728, R20
MOV  R729, R20
MOV  R730, R20
MOV  R731, R20
MOV  R732, R20
MOV  R733, R20
MOV  R734, R20
MOV  R735, R20
MOV  R736, R20
MOV  R737, R20
MOV  R738, R20
MOV  R739, R20
MOV  R740, R20
MOV  R741, R20
MOV  R742, R20
MOV  R743, R20
MOV  R744, R20
MOV  R745, R20
MOV  R746, R20
MOV  R747, R20
MOV  R748, R20
MOV  R749, R20
MOV  R750, R20
MOV  R751, R20
MOV  R752, R20
MOV  R753, R20
MOV  R754, R20
MOV  R755, R20
MOV  R756, R20
MOV  R757, R20
MOV  R758, R20
MOV  R759, R20
MOV  R760, R20
MOV  R761, R20
MOV  R762, R20
MOV  R763, R20
MOV  R764, R20
MOV  R765, R20
MOV  R766, R20
MOV  R767, R20
MOV  R768, R20
MOV  R769, R20
MOV  R770, R20
MOV  R771, R20
MOV  R772, R20
MOV  R773, R20
MOV  R774, R20
MOV  R775, R20
MOV  R776, R20
MOV  R777, R20
MOV  R778, R20
MOV  R779, R20
MOV  R780, R20
MOV  R781, R20
MOV  R782, R20
MOV  R783, R20
MOV  R784, R20
MOV  R785, R20
MOV  R786, R20
MOV  R787, R20
MOV  R788, R20
MOV  R789, R20
MOV  R790, R20
MOV  R791, R20
MOV  R792, R20
MOV  R793, R20
MOV  R794, R20
MOV  R795, R20
MOV  R796, R20
MOV  R797, R20
MOV  R798, R20
MOV  R799, R20
MOV  R800, R20
MOV  R801, R20
MOV  R802, R20
MOV  R803, R20
MOV  R804, R20
MOV  R805, R20
MOV  R806, R20
MOV  R807, R20
MOV  R808, R20
MOV  R809, R20
MOV  R810, R20
MOV  R811, R20
MOV  R812, R20
MOV  R813, R20
MOV  R814, R20
MOV  R815, R20
MOV  R816, R20
MOV  R817, R20
MOV  R818, R20
MOV  R819, R20
MOV  R820, R20
MOV  R821, R20
MOV  R822, R20
MOV  R823, R20
MOV  R824, R20
MOV  R825, R20
MOV  R826, R20
MOV  R827, R20
MOV  R828, R20
MOV  R829, R20
MOV  R830, R20
MOV  R831, R20
MOV  R832, R20
MOV  R833, R20
MOV  R834, R20
MOV  R835, R20
MOV  R836, R20
MOV  R837, R20
MOV  R838, R20
MOV  R839, R20
MOV  R840, R20
MOV  R841, R20
MOV  R842, R20
MOV  R843, R20
MOV  R844, R20
MOV  R845, R20
MOV  R846, R20
MOV  R847, R20
MOV  R848, R20
MOV  R849, R20
MOV  R850, R20
MOV  R851, R20
MOV  R852, R20
MOV  R853, R20
MOV  R854, R20
MOV  R855, R20
MOV  R856, R20
MOV  R857, R20
MOV  R858, R20
MOV  R859, R20
MOV  R860, R20
MOV  R861, R20
MOV  R862, R20
MOV  R863, R20
MOV  R864, R20
MOV  R865, R20
MOV  R866, R20
MOV  R867, R20
MOV  R868, R20
MOV  R869, R20
MOV  R870, R20
MOV  R871, R20
MOV  R872, R20
MOV  R873, R20
MOV  R874, R20
MOV  R875, R20
MOV  R876, R20
MOV  R877, R20
MOV  R878, R20
MOV  R879, R20
MOV  R880, R20
MOV  R881, R20
MOV  R882, R20
MOV  R883, R20
MOV  R884, R20
MOV  R885, R20
MOV  R886, R20
MOV  R887, R20
MOV  R888, R20
MOV  R889, R20
MOV  R890, R20
MOV  R891, R20
MOV  R892, R20
MOV  R893, R20
MOV  R894, R20
MOV  R895, R20
MOV  R896, R20
MOV  R897, R20
MOV  R898, R20
MOV  R899, R20
MOV  R900, R20
MOV  R901, R20
MOV  R902, R20
MOV  R903, R20
MOV  R904, R20
MOV  R905, R20
MOV  R906, R20
MOV  R907, R20
MOV  R908, R20
MOV  R909, R20
MOV  R910, R20
MOV  R911, R20
MOV  R912, R20
MOV  R913, R20
MOV  R914, R20
MOV  R915, R20
MOV  R916, R20
MOV  R917, R20
MOV  R918, R20
MOV  R919, R20
MOV  R920, R20
MOV  R921, R20
MOV  R922, R20
MOV  R923, R20
MOV  R924, R20
MOV  R925, R20
MOV  R926, R20
MOV  R927, R20
MOV  R928, R20
MOV  R929, R20
MOV  R930, R20
MOV  R931, R20
MOV  R932, R20
MOV  R933, R20
MOV  R934, R20
MOV  R935, R20
MOV  R936, R20
MOV  R937, R20
MOV  R938, R20
MOV  R939, R20
MOV  R940, R20
MOV  R941, R20
MOV  R942, R20
MOV  R943, R20
MOV  R944, R20
MOV  R945, R20
MOV  R946, R20
MOV  R947, R20
MOV  R948, R20
MOV  R949, R20
MOV  R950, R20
MOV  R951, R20
MOV  R952, R20
MOV  R953, R20
MOV  R954, R20
MOV  R955, R20
MOV  R956, R20
MOV  R957, R20
MOV  R958, R20
MOV  R959, R20
MOV  R960, R20
MOV  R961, R20
MOV  R962, R20
MOV  R963, R20
MOV  R964, R20
MOV  R965, R20
MOV  R966, R20
MOV  R967, R20
MOV  R968, R20
MOV  R969, R20
MOV  R970, R20
MOV  R971, R20
MOV  R972, R20
MOV  R973, R20
MOV  R974, R20
MOV  R975, R20
MOV  R976, R20
MOV  R977, R20
MOV  R978, R20
MOV  R979, R20
MOV  R980, R20
MOV  R981, R20
MOV  R982, R20
MOV  R983, R20
MOV  R984, R20
MOV  R985, R20
MOV  R986, R20
MOV  R987, R20
MOV  R988, R20
MOV  R989, R20
MOV  R990, R20
MOV  R991, R20
MOV  R992, R20
MOV  R993, R20
MOV  R994, R20
MOV  R995, R20
MOV  R996, R20
MOV  R997, R20
MOV  R998, R20
MOV  R999, R20
MOV  R1000, R20
MOV  R1001, R20
MOV  R1002, R20
MOV  R1003, R20
MOV  R1004, R20
MOV  R1005, R20
MOV  R1006, R20
MOV  R1007, R20
MOV  R1008, R20
MOV  R1009, R20
MOV  R1010, R20
MOV  R1011, R20
MOV  R1012, R20
MOV  R1013, R20
MOV  R1014, R20
MOV  R1015, R20
MOV  R1016, R20
MOV  R1017, R20
MOV  R
```

### 3.8. EXAMPLE-1 PROGRAMMING THE SERIAL I/O CHANNEL

MCS-96 MACRO ASSEMBLER SERIAL PORT DEMO PROGRAM

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F1:SPX.SRC

OBJECT FILE: :F1:SPX.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB DEBUG

```

ERR LOC OBJECT LINE SOURCE STATEMENT
1 $TITLE('SERIAL PORT DEMO PROGRAM')
2 $PAGELENGTH(95)
3
4 ; This program initializes the serial port and echos any
5 ; character sent to it.
6
7
8 BAUD_REG equ 0EH
9 SPCON equ 11H
10 SPSTAT equ 11H
11 IOC1 equ 16H
12 IOC0 equ 15H
13 SBUF equ 07H
14 INT_PENDING equ 09H
15 SP equ 18H
16
17
18 rseg
19
20 CHR: dsb 1
21 TEMP0: dsb 1
22 TEMP1: dsb 1
23 RCV_FLAG: dsb 1
24
25
26 cseg
27
28 LD SP, #0B0H
29
30 LDB IOC1, #00100000B ; Set P2.0 to TXD
31
32 ; Baud rate = input frequency / (64*baud_val)
33 ; baud_val = (input frequency/64) / baud rate
34
35 baud_val equ 39 ; 2400 baud at 6.0 MHz
36
37
38 BAUD_HIGH equ ((baud_val-1)/256) OR 80H ; Set MSB to 1
39 BAUD_LOW equ (baud_val-1) MOD 256
40
41
42 LDB BAUD_REG, #BAUD_LOW
43 LDB BAUD_REG, #BAUD_HIGH
44
45 LDB SPCON, #01001001B ; Enable receiver, Mode 1
46
47 ; The serial port is now initialized
48
49
50 STB SBUF, CHR ; Clear serial Port
51 LDB TEMP0, #00100000B ; Set TI-temp
52
53 wait: JBC INT_PENDING, 6, wait ; Wait for pending bit to be set
54 ANDB INT_PENDING, #10111111B ; Clear pending bit
55
56 ORB TEMP0, SPCON ; Put SPCON into temp register
57 ; This is necessary because reading
58 ; SPCON clears TI and RI
59
60 get_byte:
61 JBC TEMP0, 6, put_byte ; If RI-temp is not set
62 STB SBUF, CHR ; Store byte
63 ANDB TEMP0, #10111111B ; CLR RI-temp
64 LDB RCV_FLAG, #0FFH ; Set bit-received flag
65
66 put_byte:
67 JBC RCV_FLAG, 0, continue ; If receive flag is cleared
68 JBC TEMP0, 5, continue ; If TI was not set
69 LDB SBUF, CHR ; Send byte
70 ANDB TEMP0, #10111111B ; CLR TI-temp
71 LDB RCV_FLAG, #00 ; Clear bit-received flag
72
73 continue:
74 BR wait
75
76 END

```

ASSEMBLY COMPLETED, NO ERROR(S) FOUND.

# WITH THE HSO UNIT

SERIAL NO CHANNEL

MCS-96 MACRO ASSEMBLER HSO EXAMPLE PROGRAM FOR PWM OUTPUTS

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F1:HSO2K.SRC

OBJECT FILE: :F1:HSO2K.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB DEBUG

```

ERR LOC OBJECT LINE SOURCE STATEMENT
1 $TITLE ('HSO EXAMPLE PROGRAM FOR PWM OUTPUTS')
2 $PAGELENGTH(95)
3
4 ; This program will provide 4 PWM outputs on HSO pins 0-3
5 ; The input parameters passed to the program are:
6 ;
7 ; HSO_ON N HSO on time for pin N
8 ; HSO_OFF N HSO off time for pin N
9
10 ; Where: Times are in timer1 cycles
11 ; N takes values from 0 to 3
12
13 ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
14
15
16 dseg
17
18 D_STAT: DSB 1
19 extrn HSO_ON_0:word, HSO_OFF_0:word
20 extrn HSO_ON_1:word, HSO_OFF_1:word
21 extrn HSO_ON_2:word, HSO_OFF_2:word
22 extrn HSO_ON_3:word, HSO_OFF_3:word
23 extrn HSO_TIME:word, HSO_COMMAND:byte
24 extrn TIMER1:word, IOS0:byte
25 extrn SP:word
26
27
28 rseg
29
30 public OLD_STAT
31 OLD_STAT: dsb 1
32 NEW_STAT: dsb 1
33
34
35 cseg
36
37 PUBLIC wait
38
39
40 wait: JBS IOS0, 6, wait ; Loop until HSO holding register
41 NOP ; is empty
42
43 STB IOS0, D_STAT ; Load byte to external RAM
44
45 ; For operation with interrupts 'store_stat:' would be the
46 ; entry point of the routine.
47 ; Note that a DI or PUSHF might have to be added.
48
49 store_stat:
50 ANDB NEW_STAT, IOS0, #0FH ; Store new status of HSO
51 CMPB OLD_STAT, NEW_STAT
52 JE wait ; If status hasn't changed
53 XORB OLD_STAT, NEW_STAT
54
55
56 check_0:
57 JBC OLD_STAT, 0, check_1 ; Jump if OLD_STAT(0)=NEW_STAT(0)
58 JBS NEW_STAT, 0, set_off_0
59
60 set_on_0:
61 LDB HSO_COMMAND, #00110000B ; Set HSO for timer1, set pin 0
62 ADD HSO_TIME, TIMER1, HSO_OFF_0 ; Time to set pin = timer1 value
63 BR check_1 ; + Time for pin to be low
64
65 set_off_0:
66 LDB HSO_COMMAND, #00010000B ; Set HSO for timer1, clear pin 0
67 ADD HSO_TIME, TIMER1, HSO_ON_0 ; Time to clear pin = timer1 value
68 ; + Time for pin to be high
69
70
71 check_1:
72 JBC OLD_STAT, 1, check_2 ; Jump if OLD_STAT(1)=NEW_STAT(1)
73 JBS NEW_STAT, 1, set_off_1
74
75 set_on_1:
76 LDB HSO_COMMAND, #00110001B ; Set HSO for timer1, set pin 1
77 ADD HSO_TIME, TIMER1, HSO_OFF_1 ; Time to set pin = timer1 value
78 BR check_2 ; + Time for pin to be low
79
80 set_off_1:
81 LDB HSO_COMMAND, #00010001B ; Set HSO for timer1, clear pin 1
82 ADD HSO_TIME, TIMER1, HSO_ON_1 ; Time to clear pin = timer1 value
83 ; + Time for pin to be high
84
85 $EJECT

```



# MCS-96 SOFTWARE DESIGN INFORMATION

MCS-96 MACRO ASSEMBLER HSO EXAMPLE PROGRAM FOR PWM OUTPUTS

| ERR | LOC               | OBJECT | LINE | SOURCE STATEMENT  |
|-----|-------------------|--------|------|---|
|     |                   |        | 85   |   |
|     | 004A              |        | 86   | check_2:  |
|     | 004A 320017       | R      | 87   | JBC OLD_STAT, 2, check_3 ; Jump if OLD_STAT(2)=NEW_STAT(2)        |
|     | 004D 3A010B       | R      | 88   | JBS NEW_STAT, 2, set_off_2  |
|     |                   |        | 89   |   |
|     | 0050              |        | 90   | set_on_2:   |
|     | 0050 B13200       | E      | 91   | LDB HSO_COMMAND, #00110010B ; Set HSO for timer1, set pin 2       |
|     | 0053 470100000000 | E      | 92   | ADD HSO_TIME, TIMER1, HSO_OFF_2 ; Time to set pin = Timer1 value  |
|     | 0059 2009         |        | 93   | BR check_3 ; + Time for pin to be low                             |
|     |                   |        | 94   |   |
|     | 005B              |        | 95   | set_off_2:  |
|     | 005B B11200       | E      | 96   | LDB HSO_COMMAND, #00010010B ; Set HSO for timer1, clear pin 2     |
|     | 005E 470100000000 | E      | 97   | ADD HSO_TIME, TIMER1, HSO_ON_2 ; Time to clear pin = Timer1 value |
|     |                   |        | 98   |   |
|     |                   |        | 99   |   |
|     |                   |        | 100  |   |
|     | 0064              |        | 101  | check_3:  |
|     | 0064 320017       | R      | 102  | JBC OLD_STAT, 3, check_done ; Jump if OLD_STAT(3)=NEW_STAT(3)     |
|     | 0067 3B010B       | R      | 103  | JBS NEW_STAT, 3, set_off_3  |
|     |                   |        | 104  |   |
|     | 006A              |        | 105  | set_on_3:   |
|     | 006A B13300       | E      | 106  | LDB HSO_COMMAND, #00110011B ; Set HSO for timer1, set pin 3       |
|     | 006D 470100000000 | E      | 107  | ADD HSO_TIME, TIMER1, HSO_OFF_3 ; Time to set pin = Timer1 value  |
|     | 0073 2009         |        | 108  | BR check_done ; + Time for pin to be low                          |
|     |                   |        | 109  |   |
|     | 0075              |        | 110  | set_off_3:  |
|     | 0075 B11300       | E      | 111  | LDB HSO_COMMAND, #00010011B ; Set HSO for timer1, clear pin 3     |
|     | 0078 470100000000 | E      | 112  | ADD HSO_TIME, TIMER1, HSO_ON_3 ; Time to clear pin = Timer1 value |
|     |                   |        | 113  |   |
|     |                   |        | 114  |   |
|     |                   |        | 115  |   |
|     | 007E              |        | 116  | check_done:   |
|     | 007E B00100       | R      | 117  | LDB OLD_STAT, NEW_STAT ; Store current status and                 |
|     |                   |        | 118  |   |
|     | 0081 F0           |        | 119  | RET ; wait for interrupt flag                                     |
|     |                   |        | 120  |   |
|     |                   |        | 121  |   |
|     | 0082              |        | 122  | END   |

ASSEMBLY COMPLETED, NO ERROR(S) FOUND.

### 3.10. EXAMPLE-3 MEASURING PULSES WITH THE HSI UNIT

MCS-96 MACRO ASSEMBLER MEASURING PULSES USING THE HSI UNIT

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: \*F1:PULSEX.SRC  
OBJECT FILE: \*F1:PULSEX.OBJ  
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB DEBUG

```

ERR LOC OBJECT LINE SOURCE STATEMENT
1 $TITLE('MEASURING PULSES USING THE HSI UNIT')
2 $PAGELENGTH(95)
3
4 ; This program measures pulsewidths in TIMER1 cycles
5 ; and returns the values in external RAM.
6
7
8 SP equ 18H
9 HSI_MODE equ 03H
10 HSI_STATUS equ 06H
11 HSI_TIME equ 0AH
12 TIMER1 equ 0AH
13 IOC0 equ 15H
14 IOS1 equ 16H
15
16
17 rseg
18
19 HIGH TIME: dsw 1
20 LOW TIME: dsw 1
21 PERIOD: dsw 1
22 HI_EDGE: dsw 1
23 LO_EDGE: dsw 1
24
25
26 AX: dsw 1
27 AL equ AX :byte
28 AH equ (AX+1) :byte
29
30 BX: dsw 1
31 BL equ BX :byte
32 BU equ (BX+1) :byte ; Note that 'BH' is an opcode so it
33 ; can't be used as a label
34
35 cseg
36
37 LD SP, #0C0H
38 LDB IOC0, #00000001B ; Enable HSI 0
39 LDB HSI_MODE, #00001111B ; HSI 0 look for either edge
40
41 wait: ADD PERIOD, HIGH_TIME, LOW_TIME
42
43 JBC IOS1, 7, wait ; Wait while no pulse is entered
44
45 LDB AL, HSI_STATUS ; Load status; Note that reading
46 ; HSI_TIME clears HSI_STATUS
47
48 LD BX, HSI_TIME ; Load the HSI_TIME
49
50 JBS AL, 1, hsi_hi ; Jump if HSI.0 is high
51
52 hsi_lo: ST BX, LO_EDGE
53 SUB HIGH_TIME, LO_EDGE, HI_EDGE
54 BR wait
55
56
57 hsi_hi: ST BX, HI_EDGE
58 SUB LOW_TIME, HI_EDGE, LO_EDGE
59 BR wait
60
61 END

```

ASSEMBLY COMPLETED, NO ERROR(S) FOUND.

## 3.11. EXAMPLE-4 SCANNING THE A/D CHANNELS

MCS-96 MACRO ASSEMBLER SCANNING THE A TO D CHANNELS

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F1:ATODX.SRC

OBJECT FILE: :F1:ATODX.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB DEBUG

```

ERR LOC OBJECT LINE SOURCE STATEMENT
1 ($TITLE('SCANNING THE A TO D CHANNELS'))
2 $PAGELength(95)
3
4 This program scans A to D lines 0 through 3 and stores the
5 results in RESULT_N
6
7 AD_RESULT LO equ 02
8 AD_RESULT HI equ 03
9 AD_COMMAND equ 02
10 SP equ 18H
11
12
13 dseg
14
15 RESULT TABLE:
16 RESULT_1: dsw 1
17 RESULT_2: dsw 1
18 RESULT_3: dsw 1
19 RESULT_4: dsw 1
20
21 rseg
22
23 AX: dsw 1
24 AL equ AX :byte
25 AH equ (AX+1) :byte
26
27 BX: dsw 1
28 BL equ BX :byte
29 BU equ (BX+1) :byte
30
31
32
33 DX: dsw 1
34 DL equ DX :byte
35 DH equ (DX+1) :byte
36
37
38 cseg
39
40
41 start: LD SP, #0C0H ; Set Stack Pointer
42 LD BX, 00H ; Use the zero register
43
44 next: ORB BL, #1000B ; Start conversion on channel
45 LDB AD_COMMAND, BL ; indicated by BL register
46 ANDB BL, #0111B
47
48 NOP ; Wait for conversion to start
49
50 check: JBS AD_RESULT_LO, 3, check ; Wait while A to D is busy
51
52
53 LDB AL, AD_RESULT_LO ; Load low order result
54 LDB AH, AD_RESULT_HI ; Load high order result
55
56 ADDB DL, BL, BL ; DL=BL*2
57 LDBZ DX, DL
58 ST AX, RESULT_TABLE[DX] ; Store result indexed by BL*2
59
60 INCB BL
61 ANDB BL, #03H
62
63 BR next
64 END

```

ASSEMBLY COMPLETED, NO ERROR(S) FOUND.

3.12. EXAMPLE-5 TABLE LOOKUP-AND  
INTERPOLATION

MCS-96 MACRO ASSEMBLER TABLE LOOKUP AND INTERPOLATION

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F1:INTERX.SRC

OBJECT FILE: :F1:INTERX.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB DEBUG

```

ERR LOC OBJECT          LINE    SOURCE STATEMENT
1      $TITLE('TABLE LOOKUP AND INTERPOLATION')
2      $PAGELENGTH(95)
3
4      ; This program uses a lookup table to generate 12-bit function values
5      ; using 8-bit input values. The table is 16 bytes long and 16 bits wide.
6      ; A linear interpolation is made using the following formula:
7
8      ;      Table Step = (IN_VAL - TABLE_LOW) * (TABLE_HIGH - TABLE_LOW) / (OUT - TABLE_LOW)
9      ;
10     ;      TABLE_HIGH - TABLE_LOW      OUT - TABLE_LOW
11     ;
12     ;      16      IN_DIF
13     ;      ----- = -----
14     ;      TAB_DIF  OUT_DIF
15
16     ; Cross Multiplication is used to solve for OUT_DIF
17
18
19
20     0018 SP      equ      18H
21
22
23     0000 dseg
24
25     0000 RESULT TABLE:
26     0000 RESULT:      dsb      1
27
28
29     0000 rseg
30
31     0000 AX:      dsb      1
32     0000 AL      equ      AX      :byte
33     0001 AH      equ      (AX+1) :byte
34
35     0002 BX:      dsb      1
36     0002 BL      equ      BX      :byte
37     0003 BU      equ      (BX+1) :byte ; Note that 'BH' is an opcode so it
; can't be used as a label
38
39
40     0004 IN_VAL:      dsb      1
41     0006 TABLE_LOW: dsb      1
42     0008 TABLE_HIGH: dsb      1
43     000A IN_DIF:      dsb      1
44     000C IN_DIFB      equ      IN_DIF :byte
45     000E TAB_DIF:      dsb      1
46     0010 OUT:      dsb      1
47     OUT_DIF:      dsb      1
48
49
50     0000 cseg
51
52
53     0000 A1C00018 53 start: LD      SP, #0C0H ; Set Stack Pointer
54
55
56     0004 B00400 R 56 look: LDB AL, IN_VAL
57     0007 180300 R 57 SHRB AL, #3 ; Place 2 times the upper nibble in byte
58     000A 71FE00 R 58 ANDB AL, #11111110B ; Insure AL is a word address
59     000D AC0000 R 59 LDBZE AX, AL
60
61     0010 A300420006 R 61 LD TABLE_LOW, TABLE [AX] ; TABLE_LOW is table output value
62 ; of IN_VAL rounded down to the
63 ; nearest multiple of 10H.
64
65     0015 A300440008 R 65 LD TABLE_HIGH, TABLE+2 [AX] ; TABLE_HIGH is the table output
66 ; value of IN_VAL rounded up to the
67 ; nearest multiple of 10H.
68
69
70     001A 4806080C R 70 SUB TAB_DIF, TABLE_HIGH, TABLE_LOW
71
72     001E 510F040A R 72 ANDB IN_DIFB, IN_VAL, #0FH
73     0022 BC0A0A R 73 LDBSE IN_DIF, IN_DIFB ; Make input difference into a word
74
75     0025 FE4C0C0A10 R 75 MUL OUT_DIF, IN_DIF, TAB_DIF
76     002A FE8D100010 R 76 DIV OUT_DIF, #16
77
78     002F 4406100E R 78 lab1: ADD OUT_DIF, TABLE_LOW ; Add output difference to output
79 ; generated with truncated IN_VAL
80 ; as input
81     0033 08040E R 81 SHR OUT, #4 ; Round to 12-bit answer

```



# MCS-96 SOFTWARE DESIGN INFORMATION

MCS-96 MACRO ASSEMBLER

TABLE LOOKUP AND INTERPOLATION

10/11/83

```
ERR LOC OBJECT LINE SOURCE STATEMENT
0036 D307 82 JNC lab2
0038 070E 83 INC OUT ; Round up if Carry = 1
003A C30100000E 84 ST OUT, RESULT
003F 27C3 85 lab2: BR look
0041 86 cseg
0042 000000200034004C 87
004A 005D0006A00720078 88
0052 007B007D0076006D 89
005A 005D0004B00340022 90
0062 0010 91 table: DCW 0000H, 2000H, 3400H, 4C00H
0064 0010 92 DCW 5D00H, 6A00H, 7200H, 7800H
0064 0010 93 DCW 7B00H, 7D00H, 7600H, 6D00H
0064 0010 94 DCW 5D00H, 4B00H, 3400H, 2200H
0064 0010 95 DCW 1000H
0064 0010 96
0064 0010 97 END
```

ASSEMBLY COMPLETED, NO ERROR(S) FOUND.

Flag Settings. The modification to the flag setting is shown for each instruction. A checkmark (✓) means that the flag is set or cleared as appropriate. A hyphen means that the flag is not modified. A one or zero (1) or (0) indicates that the flag will be in that state after the instruction. An up arrow (↑) indicates that the instruction may set the flag if it is appropriate but will not clear the flag. A down arrow (↓) indicates that the flag can be cleared but not set by the instruction. A question mark (?) indicates that the flag will be left in an indeterminate state after the operation.

Generic Jumps and Calls. The assembler for the 8096 provides for generic jumps and calls. For all of the conditional jump instructions a "B" can be substituted for the "J" and the assembler will generate a code sequence which is logically equivalent but can reach anywhere in the memory. A JH can only jump about 128 locations from the current program counter; a BH can jump anywhere in memory. In a like manner a BR will cause a JMP or LJM to be generated as appropriate and a CALL will cause a SCALL or LCALL to be generated. The assembler user guide (see section 3.0) should be consulted for the algorithms used by the assembler to convert these generic instructions into actual machine instructions.

Indirect Shifts. The indirect shift operations use registers 24 through 25 (18H-0FFH), since 0-15 are direct operation registers. Note that indirect shifts through SPH are illegal operations.

The maximum shift count is 31 (1FH). Count values above this will be truncated to the 2 least significant bits.

aa. A two bit field within an opcode which selects the basic addressing mode user. This field is only present in those opcodes which allow address mode options. The encoding of the field is as follows:

| aa | Addressing mode |
|----|-----------------|
| 00 | Register direct |
| 01 | Immediate       |
| 10 | Indirect        |
| 11 | Indexed         |

The selection between indirect and indirect with auto-increment or between short and long indexing is done based on the least significant bit of the instruction byte which follows the opcode. This type selects the 16-bit register which is to be used in the address calculation. Since the 8096 requires that words be aligned on even byte boundaries this bit would be otherwise unused.

byte. A byte register in the internal register file. When confusion could exist as to whether this field refers to a source or a destination register it will be prefixed with an "S" or a "D".

brnop. A byte operand which is addressed by any of the address modes discussed in section 3.2.

bitno. A three bit field within an instruction opcode which selects one of the eight bits in a byte.

wreg. A word register in the internal register file. When confusion could exist as to whether this field refers to a source register or a destination register it will be prefixed with an "S" or a "D".

### 3.13. DETAILED INSTRUCTION SET DESCRIPTION

This section gives a description of each instruction recognized by the 8096 sorted alphabetically by the mnemonic used in the assembly language for the 8096. Note that the effect on the program counter (PC) is not always shown in the instruction descriptions. All instructions increment the PC by the number of bytes in the instruction. Several acronyms are used in the instruction set descriptions which are defined here:

**aa.** A two bit field within an opcode which selects the basic addressing mode user. This field is only present in those opcodes which allow address mode options. The encoding of the field is as follows:

| aa | Addressing mode |
|----|-----------------|
| 00 | Register direct |
| 01 | Immediate       |
| 10 | Indirect        |
| 11 | Indexed         |

The selection between indirect and indirect with auto-increment or between short and long indexing is done based on the least significant bit of the instruction byte which follows the opcode. This type selects the 16-bit register which is to take part in the address calculation. Since the 8096 requires that words be aligned on even byte boundaries this bit would be otherwise unused.

**breg.** A byte register in the internal register file. When confusion could exist as to whether this field refers to a source or a destination register it will be prefixed with an "S" or a "D."

**baop.** A byte operand which is addressed by any of the address modes discussed in section 3.2.

**bitno.** A three bit field within an instruction op-code which selects one of the eight bits in a byte.

**wreg.** A word register in the internal register file. When confusion could exist as to whether this field refers to a source register or a destination register it will be prefixed with an "S" or a "D."

**waop.** A word operand which is addressed by any of the address modes discussed in section 3.2.

**Lreg.** A 32-bit register in the internal register file.

**BEA.** Extra bytes of code required for the address mode selected.

**CEA.** Extra state times (cycles) required for the address mode selected.

**cadd** An address in the program code.

**Flag Settings.** The modification to the flag setting is shown for each instruction. A checkmark (✓) means that the flag is set or cleared as appropriate. A hyphen means that the flag is not modified. A one or zero (1) or (0) indicates that the flag will be in that state after the instruction. An up arrow (↑) indicates that the instruction may set the flag if it is appropriate but will not clear the flag. A down arrow (↓) indicates that the flag can be cleared but not set by the instruction. A question mark (?) indicates that the flag will be left in an indeterminate state after the operation.

**Generic Jumps and Calls.** The assembler for the 8096 provides for generic jumps and calls. For all of the conditional jump instructions a "B" can be substituted for the "J" and the assembler will generate a code sequence which is logically equivalent but can reach anywhere in the memory. A JH can only jump about 128 locations from the current program counter; a BH can jump anywhere in memory. In a like manner a BR will cause a SJMP or LJMP to be generated as appropriate and a CALL will cause a SCALL or LCALL to be generated. The assembler user guide (see section 3.0) should be consulted for the algorithms used by the assembler to convert these generic instructions into actual machine instructions.

**Indirect Shifts.** The indirect shift operations use registers 24 through 255 (18H-0FFH), since 0-15 are direct operators and registers 16 through 23 are Special Function Registers. Note that indirect shifts through SFRs are illegal operations.

The maximum shift count is 31 (1FH). Count values above this will be truncated to the 5 least significant bits.

### 3.13.1. ADD (Two Operands) — ADD WORDS

**Operation:** The sum of the two word operands is stored into the destination (leftmost) operand.

$$(DEST) \leftarrow (DEST) + (SRC)$$

**Assembly Language Format:**      DST      SRC  
ADD    wreg,    waop

**Object Code Format:** [ 011001aa ][ waop ][ wreg ]

Bytes: 2 + BEA  
States: 4 + CEA

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| ✓              | ✓ | ✓ | ✓ | ↑  | —  |

### 3.13.2. ADD (Three Operands) — ADD WORDS

**Operation:** The sum of the second and third word operands is stored into the destination (leftmost) operand.

$$(DEST) \leftarrow (SRC1) + (SRC2)$$

**Assembly Language Format:**      DST      SRC1      SRC2  
ADD    Dwreg, Swreg,    waop

**Object Code Format:** [ 010001aa ][ waop ][ Swreg ][ Dwreg ]

Bytes: 3 + BEA  
States: 5 + CEA

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| ✓              | ✓ | ✓ | ✓ | ↑  | —  |

**3.13.3. ADDB (Two Operands) — ADD BYTES**

**Operation:** The sum of the two byte operands is stored into the destination (leftmost) operand.

$$(DEST) \leftarrow (DEST) + (SRC)$$

**Assembly Language Format:**

DST SRC  
ADDB breg, baop

**Object Code Format:** [ 01101aa ][ baop ][ breg ]

Bytes: 2 + BEA

States: 4 + CEA

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| ✓              | ✓ | ✓ | ✓ | ↑  | -  |

**3.13.4. ADDB (Three Operands) — ADD BYTES**

**Operation:** The sum of the second and third byte operands is stored into the destination (leftmost) operand.

$$(DEST) \leftarrow (SRC1) + (SRC2)$$

**Assembly Language Format:**

DST SRC1 SRC2  
ADDB Dbreg, Sbreg, baop

**Object Code Format:** [ 010101aa ][ baop ][ Sbreg ][ Dbreg ]

Bytes: 3 + BEA

States: 5 + CEA

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| ✓              | ✓ | ✓ | ✓ | ↑  | -  |



**3.13.5. ADDC — ADD WORDS WITH CARRY**

**Operation:** The sum of the two word operands and the carry flag (0 or 1) is stored into the destination (leftmost) operand.

$$(\text{DEST}) \leftarrow (\text{DEST}) + (\text{SRC}) + \text{C}$$

**Assembly Language Format:**      DST      SRC  
 ADDC    wreg,    waop

**Object Code Format:** [ 101001aa ][ waop ][ wreg ]

Bytes: 2+BEA

States: 4+BEA

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| ↓              | ✓ | ✓ | ✓ | ↑  | —  |

**3.13.6. ADDCB — ADD BYTES WITH CARRY**

**Operation:** The sum of the two byte operands and the carry flag (0 or 1) is stored into the destination (leftmost) operand.

$$(\text{DEST}) \leftarrow (\text{DEST}) + (\text{SRC}) + \text{C}$$

**Assembly Language Format:**      DST      SRC  
 ADDCB    breg,    baop

**Object Code Format:** [ 101101aa ][ baop ][ breg ]

Bytes: 2+BEA

States: 4+CEA

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| ↓              | ✓ | ✓ | ✓ | ↑  | —  |

**3.13.7. AND (Two Operands) — LOGICAL AND WORDS**

**Operation:** The two word operands are ANDed, the result having a 1 only in those bit positions where both operands had a 1, with zeroes in all other bit positions. The result is stored into the destination (leftmost) operand.

(DEST) ← (DEST) AND (SRC)

**Assembly Language Format:**            DST    SRC  
AND    wreg,    waop

**Object Code Format:** [ 011000aa ][ waop ][ wreg ]

Bytes: 2 + BEA

States: 4 + CEA

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| ✓              | ✓ | 0 | 0 | —  | —  |

**3.13.8. AND (Three Operands) — LOGICAL AND WORDS**

**Operation:** The second and third word operands are ANDed, the result having a 1 only in those bit positions where both operands had a 1, with zeroes in all other bit positions. The result is stored into the destination (leftmost) operand.

(DEST) ← (SRC1) AND (SRC2)

**Assembly Language Format:**            DST    SRC1    SRC2  
AND    Dwreg,    Swreg,    waop

**Object Code Format:** [ 010000aa ][ waop ][ Swreg ][ Dwreg ]

Bytes: 3 + BEA

States: 5 + CEA

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| ✓              | ✓ | 0 | 0 | —  | —  |

**3.13.9. ANDB (Two Operands) — LOGICAL AND BYTES**

**Operation:** The two byte operands are ANDed, the result having a 1 only in those bit positions where both operands had a 1, with zeroes in all other bit positions. The result is stored into the destination (leftmost) operand.

$(\text{DEST}) \leftarrow (\text{DEST}) \text{ AND } (\text{SRC})$

**Assembly Language Format:**

DST SRC  
ANDB breg, baop

**Object Code Format:** [ 011100aa ][ baop ][ breg ]

Bytes: 2 + BEA

States: 4 + CEA

Flags Affected

| Z | N | C | V | VT | ST |
|---|---|---|---|----|----|
| ✓ | ✓ | 0 | 0 | —  | —  |

**3.13.10. ANDB (Three Operands) — LOGICAL AND BYTES**

**Operation:** The second and third byte operands are ANDed, the result having a 1 only in those bit positions where both operands had a 1, with zeroes in all other bit positions. The result is stored into the destination (leftmost) operand.

$(\text{DEST}) \leftarrow (\text{SRC1}) \text{ AND } (\text{SRC2})$

**Assembly Language Format:**

DST SRC1 SRC2  
ANDB Dbreg, Sbreg, baop

**Object Code Format:** [ 010100aa ][ baop ][ Sbreg ][ Dbreg ]

Bytes: 3 + BEA

States: 5 + CEA

Flags Affected

| Z | N | C | V | VT | ST |
|---|---|---|---|----|----|
| ✓ | ✓ | 0 | 0 | —  | —  |

**Operation:** The execution continues at the address specified in the operand word register.  
 $PC \leftarrow (DEST)$

**Assembly Language Format:** BR [wreg]

**Object Code Format:** [ 11100011 ] [wreg]

Bytes: 2

States: 8

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| -              | - | - | - | -  | -  |

### 3.13.12. CLR — CLEAR WORD

**Operation:** The value of the word operand is set to zero.

$(DEST) \leftarrow 0$

**Assembly Language Format:** CLR wreg

**Object Code Format:** [ 00000001 ] [ wreg ]

Bytes: 2

States: 4

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| 1              | 0 | 0 | 0 | -  | -  |



**3.13.13. CLRB — CLEAR BYTE****Operation:** The value of the byte operand is set to zero.(DEST)  $\leftarrow 0$ **Assembly Language Format:** CLRB breg**Object Code Format:** [ 00010001 ] [ breg ]

Bytes: 2

States: 4

Flags Affected

| Z | N | C | V | VT | ST |
|---|---|---|---|----|----|
| 1 | 0 | 0 | 0 | —  | —  |

**3.13.14. CLRC — CLEAR CARRY FLAG****Operation:** The value of the carry flag is set to zero. $C \leftarrow 0$ **Assembly Language Format:** CLRC**Object Code Format:** [ 11111000 ]

Bytes: 1

States: 4

Flags Affected

| Z | N | C | V | VT | ST |
|---|---|---|---|----|----|
| — | — | 0 | — | —  | —  |

**3.13.15. CLRVT — CLEAR OVERFLOW TRAP**

**Operation:** The value of the overflow-trap flag is set to zero.

$VT \leftarrow 0$

**Assembly Language Format:** CLRVT

**Object Code Format:** [ 11111100 ]

Bytes: 1

States: 4

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| —              | — | — | — | 0  | —  |

**3.13.16. CMP — COMPARE WORDS**

**Operation:** The source (rightmost) word operand is subtracted from the destination (leftmost) word operand. The flags are altered but the operands remain unaffected. The carry flag is set as complement of borrow.

(DEST) — (SRC)

**Assembly Language Format:**           DST     SRC  
CMP   wreg,   waop

**Object Code Format:** [ 100010aa ][ waop ][ wreg ]

Bytes: 2+BEA

States: 4+CEA

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| ✓              | ✓ | ✓ | ✓ | ↑  | —  |

### 3.13.17. CMPB — COMPARE BYTES

**Operation:** The source (rightmost) byte operand is subtracted from the destination (leftmost) byte operand. The flags are altered but the operands remain unaffected. The carry flag is set as complement of borrow.

(DEST) ← (SRC)

**Assembly Language Format:**            DST    SRC  
CMPB    breg,    baop

**Object Code Format:** [ 100110aa ][ baop ][ breg ]

Bytes: 2+BEA

States: 4+CEA

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| ✓              | ✓ | ✓ | ✓ | ↑  | —  |

### 3.13.18. DEC — DECREMENT WORD

**Operation:** The value of the word operand is decremented by one.

(DEST) ← (DEST) — 1

**Assembly Language Format:** DEC    wreg

**Object Code Format:** [ 00000101 ][ wreg ]

Bytes: 2

States: 4

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| ✓              | ✓ | ✓ | ✓ | ↑  | —  |

**3.13.19. DECB — DECREMENT BYTE**

**Operation:** The value of the byte operand is decremented by one.

$$(DEST) \leftarrow (DEST) - 1$$

**Assembly Language Format:** DECB breg

**Object Code Format:** [ 00010101 ][ breg ]

Bytes: 2

States: 4

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| ✓              | ✓ | ✓ | ✓ | ↑  | —  |

**3.13.20. DI — DISABLE INTERRUPTS**

**Operation:** Interrupts are disabled. Interrupt-calls will not occur after this instruction.

Interrupt Enable (PSW.9) ← 0

**Assembly Language Format:** DI

**Object Code Format:** [ 11111010 ]

Bytes: 1

States: 4

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| —              | — | — | — | —  | —  |



## 3.13.21. DIV — DIVIDE INTEGERS

**Operation:** This instruction divides the contents of the destination LONG-INTEGER operand by the contents of the INTEGER word operand, using signed arithmetic. The low order word of the destination (i.e., the word with the lower address) will contain the quotient; the high order word will contain the remainder.

$$(\text{low word DEST}) \leftarrow (\text{DEST}) / (\text{SRC})$$

$$(\text{high word DEST}) \leftarrow (\text{DEST}) \text{ MOD } (\text{SRC})$$

The above two statements are performed concurrently.

**Assembly Language Format:**           DST   SRC  
DIV   lreg,   waop

**Object Code Format:** [ 11111110 ][ 100011aa ][ waop ][ lreq ]

Bytes: 2 + BEA

States: 29 + CEA

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| -              | - | - | ? | ↑  | -  |

## 3.13.22. DIVB — DIVIDE SHORT-INTEGERS

**Operation:** This instruction divides the contents of the destination INTEGER operand by the contents of the source SHORT-INTEGER operand, using signed arithmetic. The low order byte of the destination (i.e., the byte with the lower address) will contain the quotient; the high order byte will contain the remainder.

$$(\text{low byte DEST}) \leftarrow (\text{DEST}) / (\text{SRC})$$

$$(\text{high byte DEST}) \leftarrow (\text{DEST}) \text{ MOD } (\text{SRC})$$

The above two statements are performed concurrently.

**Assembly Language Format:**           DST   SRC  
DIVB   wreg,   baop

**Object Code Format:** [ 11111110 ][ 100111aa ][ baop ][ wreg ]

Bytes: 2 + BEA

States: 21 + CEA

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| -              | - | - | ? | ↑  | -  |

## 3.13.23. DIVU — DIVIDE WORDS

**Operation:** This instruction divides the content of the destination DOUBLE-WORD operand by the contents of the source WORD operand, using unsigned arithmetic. The low order word will contain the quotient; the high order WORD will contain the remainder.

(low word DEST)  $\leftarrow$  (DEST)/(SRC)

(high word DEST)  $\leftarrow$  (DEST) MOD (SRC)

The above two statements are performed concurrently.

**Assembly Language Format:**

DST SRC  
DIVU lreg, waop

**Object Code Format:** [ 100011aa ][ waop ][ lreg ]

Bytes: 2 + BEA

States: 25 + CEA

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| -              | - | - | ✓ | ↑  | -  |

## 3.13.24. DIVUB — DIVIDE BYTES

**Operation:** This instruction divides the contents of the destination WORD operand by the contents of the source BYTE operand, using unsigned arithmetic. The low order byte of the destination (i.e., the byte with the lower address) will contain the quotient; the high order byte will contain the remainder.

(low byte DEST)  $\leftarrow$  (DEST) / (SRC)

(high byte DEST)  $\leftarrow$  (DEST) MOD (SRC)

The above two statements are performed concurrently.

**Assembly Language Format:**

DST SRC  
DIVUB wreg, baop

**Object Code Format:** [ 100111aa ][ baop ][ wreg ]

Bytes: 2 + BEA

States: 17 + CEA

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| -              | - | - | ✓ | ↑  | -  |

**3.13.25. DJNZ — DECREMENT AND JUMP IF NOT ZERO**

**Operation:** The value of the byte operand is decremented by 1. If the result is not equal to 0, the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If the result of the decrement is zero then control passes to the next sequential instruction.

(COUNT)  $\leftarrow$  (COUNT) - 1

if (COUNT)  $\neq$  0 then

PC  $\leftarrow$  PC + disp (sign-extended to 16 bits)

end\_if

**Assembly Language Format:** DJNZ breg,cadd

**Object Code Format:** [ 11100000 ][ breg ][ disp ]

Bytes: 3

States: Jump Not Taken: 5

Jump Taken: 9

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| -              | - | - | - | -  | -  |

**3.13.26. EI — ENABLE INTERRUPTS**

**Operation:** Interrupts are enabled following the execution of the next statement. Interrupt-calls cannot occur immediately following this instruction.

Interrupt Enable (PSW.9)  $\leftarrow$  1

**Assembly Language Format:** EI

**Object Code Format:** [ 11111011 ]

Bytes: 1

States: 4

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| -              | - | - | - | -  | -  |

**3.13.27. EXT — SIGN EXTEND INTEGER INTO LONG-INTEGER**

**Operation:** The low order word of the operand is sign-extended throughout the high order word of the operand.

if (low word DEST) < 8000H then  
 (high word DEST) ← 0  
 else  
 (high word DEST) ← 0FFFFH  
 end \_ if

**Assembly Language Format:** EXT lreg

**Object Code Format:** [ 00000110 ][ lreg ]

Bytes: 2

States: 4

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| ✓              | ✓ | 0 | 0 | —  | —  |

**3.13.28. EXTB — SIGN EXTEND SHORT-INTEGER INTO INTEGER**

**Operation:** The low order byte of the operand is sign-extended throughout the high order byte of the operand.

if (low byte DEST) < 80H then  
 (high byte DEST) ← 0  
 else  
 (high byte DEST) ← 0FFH  
 end \_ if

**Assembly Language Format:** EXTB wreg

**Object Code Format:** [ 00010110 ][ wreg ]

Bytes: 2

States: 4

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| ✓              | ✓ | 0 | 0 | —  | —  |



**3.13.29. INC — INCREMENT WORD**

**Operation:** The value of the word operand is incremented by 1.

$$(DEST) \leftarrow (DEST) + 1$$

**Assembly Language Format:** INC wreg

**Object Code Format:** [ 00000111 ][ wreg ]

Bytes: 2

States: 4

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| ✓              | ✓ | ✓ | ✓ | ↑  | —  |

**3.13.30. INCB — INCREMENT BYTE**

**Operation:** The value of the byte operand is incremented by 1.

$$(DEST) \leftarrow (DEST) + 1$$

**Assembly Language Format:** INCB breg

**Object Code Format:** [ 00010111 ][ breg ]

Bytes: 2

States: 4

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| ✓              | ✓ | ✓ | ✓ | ↑  | —  |

**Operation:** The specified bit is tested. If it is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If the bit is set (i.e., 1), control passes to the next sequential instruction.

if (specified bit) = 0 then

PC ← PC + disp (sign-extended to 16 bits)

**Assembly Language Format:** JBC breg,bitno,cadd

**Object Code Format:** [ 00110bbb ][ breg ][ disp ]

where bbb is the bit number within the specified register.

Bytes: 3

States: Jump Not Taken: 5

Jump Taken: 9

Flags Affected

| Z | N | C | V | VT | ST |
|---|---|---|---|----|----|
| - | - | - | - | -  | -  |

**3.13.32. JBS — JUMP IF BIT SET**

**Operation:** The specified bit is tested. If it is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If the bit is clear (i.e., 0), control passes to the next sequential instruction.

if (specified bit) = 1 then  
 $PC \leftarrow PC + \text{disp}$  (sign-extended to 16 bits)

**Assembly Language Format:** JBS breg,bitno,cadd

**Object Code Format:** [ 00111bbb ][ breg ][ disp ]

where bbb is the bit number within the specified register.

Bytes: 3  
 States: Jump Not Taken: 5  
 Jump Taken: 9

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| -              | - | - | - | -  | -  |

**3.13.33. JC — JUMP IF CARRY FLAG IS SET**

**Operation:** If the carry flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If the carry flag is clear (i.e., 0), control passes to the next sequential instruction.

if C = 1 then  
 $PC \leftarrow PC + \text{disp}$  (sign-extended to 16 bits)

**Assembly Language Format:** JC cadd

**Object Code Format:** [ 11011011 ][ disp ]

Bytes: 2  
 States: Jump Not Taken: 4  
 Jump Taken: 8

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| -              | - | - | - | -  | -  |

**3.13.34. JE — JUMP IF EQUAL**

**Operation:** If the zero flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If the zero flag is clear (i.e., 0), control passes to the next sequential instruction.

if Z = 1 then  
 $PC \leftarrow PC + \text{disp}$  (sign-extended to 16 bits)

**Assembly Language Format:** JE cadd

**Object Code Format:** [ 11011111 ][ disp ]

Bytes: 2

States: Jump Not Taken: 4

Jump Taken: 8

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| -              | - | - | - | -  | -  |
| -              | - | - | - | -  | -  |

**3.13.35. JGE — JUMP IF SIGNED GREATER THAN OR EQUAL**

**Operation:** If the negative flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If the negative flag is set (i.e., 1), control passes to the next sequential instruction.

if N = 0 then  
 $PC \leftarrow PC + \text{disp}$  (sign-extended to 16 bits)

**Assembly Language Format:** JGE cadd

**Object Code Format:** [ 11010110 ][ disp ]

Bytes: 2

States: Jump Not Taken: 4

Jump Taken: 8

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| -              | - | - | - | -  | -  |
| -              | - | - | - | -  | -  |



**3.13.36. JGT — JUMP IF SIGNED GREATER THAN**

**Operation:** If both the negative flag and the zero flag are clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If either the negative flag or the zero flag are set (i.e., 1,) control passes to the next sequential instruction.

if  $N = 0$  AND  $Z = 0$  then  
 $PC \leftarrow PC + \text{disp}$  (sign-extended to 16 bits)

**Assembly Language Format:** JGT cadd

**Object Code Format:** [ 11010010 ][ disp ]

Bytes: 2  
 States: Jump Not Taken: 4  
 Jump Taken: 8

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| -              | - | - | - | -  | -  |

**3.13.37. JH — JUMP IF HIGHER (UNSIGNED)**

**Operation:** If the carry flag is set (i.e., 1), but the zero flag is not, the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If either the carry flag is clear or the zero flag is set, control passes to the next sequential instruction.

if  $C = 1$  and  $Z = 0$  then  
 $PC \leftarrow PC + \text{disp}$  (sign-extended to 16 bits)

**Assembly Language Format:** JH cadd

**Object Code Format:** [ 11011001 ][ disp ]

Bytes: 2  
 States: Jump Not Taken: 4  
 Jump Taken: 8

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| -              | - | - | - | -  | -  |

**3.13.38. JLE — JUMP IF SIGNED LESS THAN OR EQUAL**

**Operation:** If either the negative flag or the zero flag are set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If both the negative flag and the zero flag are clear (i.e., 0), control passes to the next sequential instruction.

if  $N = 1$  OR  $Z = 1$  then  
 $PC \leftarrow PC + \text{disp}$  (sign-extended to 16 bits)

**Assembly Language Format:** JLE cadd

**Object Code Format:** [ 11011010 ][ disp ]

Bytes: 2  
 States: Jump Not Taken: 4  
 Jump Taken: 8

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| -              | - | - | - | -  | -  |

**3.13.39. JLT — JUMP IF SIGNED LESS THAN**

**Operation:** If the negative flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If the negative flag is clear (i.e., 0), control passes to the next sequential instruction.

if  $N = 1$  then  
 $PC \leftarrow PC + \text{disp}$  (sign-extended to 16 bits)

**Assembly Language Format:** JLT cadd

**Object Code Format:** [ 11011110 ][ disp ]

Bytes: 2  
 States: Jump Not Taken: 4  
 Jump Taken: 8

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| -              | - | - | - | -  | -  |

**3.13.40. JNC — JUMP IF CARRY FLAG IS CLEAR**

**Operation:** If the carry flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If the carry flag is set (i.e., 1), control passes to the next sequential instruction.

if C = 0 then  
 $PC \leftarrow PC + \text{disp}$  (sign-extended to 16 bits)

**Assembly Language Format:** JNC cadd

**Object Code Format:** [ 11010011 ] [ disp ]

Bytes: 2  
 States: Jump Not Taken: 4  
 Jump Taken: 8

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| -              | - | - | - | -  | -  |

**3.13.41. JNE — JUMP IF NOT EQUAL**

**Operation:** If the zero flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If the zero flag is set (i.e., 1), control passes to the next sequential instruction.

if Z = 0 then  
 $PC \leftarrow PC + \text{disp}$  (sign-extended to 16 bits)

**Assembly Language Format:** JNE cadd

**Object Code Format:** [ 11010111 ] [ disp ]

Bytes: 2  
 States: Jump Not Taken: 4  
 Jump Taken: 8

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| -              | - | - | - | -  | -  |

**3.13.42. JNH — JUMP IF NOT HIGHER (UNSIGNED)**

**Operation:** If either the carry flag is clear (i.e., 0), or the zero flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If the carry flag is set (i.e., 1), or the zero flag is not, control passes to the next sequential instruction.

if  $C = 0$  OR  $Z = 1$  then

$PC \leftarrow PC + \text{disp}$  (sign-extended to 16 bits)

**Assembly Language Format:** JNH cadd

**Object Code Format:** [ 11010001 ][ disp ]

Bytes: 2

States: Jump Not Taken: 4

Jump Taken: 8

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| -              | - | - | - | -  | -  |

**3.13.43. JNST — JUMP IF STICKY BIT IS CLEAR**

**Operation:** If the sticky bit flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If the sticky bit flag is set (i.e., 1), control passes to the next sequential instruction.

if  $ST = 0$  then

$PC \leftarrow PC + \text{disp}$  (sign-extended to 16 bits)

**Assembly Language Format:** JNST cadd

**Object Code Format:** [ 11010000 ][ disp ]

Bytes: 2

States: Jump Not Taken: 4

Jump Taken: 8

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| -              | - | - | - | -  | -  |



**3.13.44. JNV — JUMP IF OVERFLOW FLAG IS CLEAR**

**Operation:** If the overflow flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If the overflow flag is set (i.e., 1), control passes to next sequential instruction.

if V = 0 then  
 $PC \leftarrow PC + \text{disp}$  (sign-extended to 16 bits)

**Assembly Language Format:** JNV cadd

**Object Code Format:** [ 11010101 ][ disp ]

Bytes: 2  
 States: Jump Not Taken: 4  
 Jump Taken: 8

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| -              | - | - | - | -  | -  |

**3.13.45. JNVT — JUMP IF OVERFLOW TRAP IS CLEAR**

**Operation:** If the overflow trap flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If the overflow trap flag is set (i.e., 1), control passes to the next sequential instruction.

if VT = 0 then  
 $PC \leftarrow PC + \text{disp}$  (sign-extended to 16 bits)

**Assembly Language Format:** JNVT cadd

**Object Code Format:** [ 11010100 ][ disp ]

Bytes: 2  
 States: Jump Not Taken: 4  
 Jumps Taken: 8

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| -              | - | - | - | 0  | -  |

### 3.13.46. JST — JUMP IF STICKY BIT IS SET

**Operation:** If the sticky bit flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of  $-128$  to  $+127$ . If the sticky bit flag is clear (i.e., 0), control passes to the next sequential instruction.

if ST = 1 then

PC ← PC + disp (sign-extended to 16 bits)

**Assembly Language Format:** JST cadd

**Object Code Format:** [ 11011000 ] [ disp ]

Bytes: 2

States: Jump Not Taken: 4

Jump Taken: 8

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| -              | - | - | - | -  | -  |

### 3.13.47. JV — JUMP IF OVERFLOW FLAG IS SET

**Operation:** If the overflow flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of  $-128$  to  $+127$ . If the overflow flag is clear (i.e., 0), control passes to the next sequential instruction.

if V = 1 then

PC ← PC + disp (sign-extended to 16 bits)

**Assembly Language Format:** JV cadd

**Object Code Format:** [ 11011101 ] [ disp ]

Bytes: 2

States: Jump Not Taken: 4

Jump Taken: 8

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| -              | - | - | - | -  | -  |

**3.13.48. JVT — JUMP IF OVERFLOW TRAP IS SET**

**Operation:** If the overflow flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of  $-128$  to  $+127$ . If the overflow trap flag is clear (i.e., 0), control passes to the next sequential instruction.

if VT = 1 then

$PC \leftarrow PC + \text{disp}$  (sign-extended to 16 bits)

**Assembly Language Format:** JVT cadd

**Object Code Format:** [ 11011100 ] [ disp ]

Bytes: 2

States: Jump Not Taken: 4

Jump Taken: 8

Flags Affected

| Z | N | C | V | VT | ST |
|---|---|---|---|----|----|
| — | — | — | — | 0  | —  |

**3.13.49. LCALL — LONG CALL**

**Operation:** The contents of the program counter (the return address) is pushed onto the stack. Then the distance from the end of this instruction to the target label is added to the program counter, effecting the call. The operand may be any address in the entire address space.

$SP \leftarrow SP - 2$

$(SP) \leftarrow PC$

$PC \leftarrow PC + \text{disp}$

**Assembly Language Format:** LCALL cadd

**Object Code Format:** [ 11101111 ] [ disp-low ] [ disp-hi ]

Bytes: 3

States: Onchip stack: 13

Offchip stack: 16

Flags Affected

| Z | N | C | V | VT | ST |
|---|---|---|---|----|----|
| — | — | — | — | —  | —  |

## 3.13.50. LD — LOAD WORD

**Operation:** The value of the source (rightmost) word operand is stored into the destination (leftmost) operand.

(DEST) ← (SRC)

**Assembly Language Format:**     DST     SRC  
LD   wreg,   waop

**Object Code Format:** [ 101000aa ][ waop ][ wreg ]

Bytes: 2 + BEA

States: 4 + CEA

Flags Affected

| Z | N | C | V | VT | ST |
|---|---|---|---|----|----|
| — | — | — | — | —  | —  |

| Z | N | C | V | VT | ST |
|---|---|---|---|----|----|
| — | — | — | — | —  | —  |

## 3.13.51. LDB — LOAD BYTE

**Operation:** The value of the source (rightmost) byte operand is stored into the destination (leftmost) operand.

(DEST) ← (SRC)

**Assembly Language Format:**     DST     SRC  
LDB   breg,   baop

**Object Code Format:** [ 101100aa ][ baop ][ breg ]

Bytes: 2 + BEA

States: 4 + CEA

Flags Affected

| Z | N | C | V | VT | ST |
|---|---|---|---|----|----|
| — | — | — | — | —  | —  |

| Z | N | C | V | VT | ST |
|---|---|---|---|----|----|
| — | — | — | — | —  | —  |



**3.13.52. LDBSE — LOAD INTEGER WITH SHORT-INTEGER**

**Operation:** The value of the source (rightmost) byte operand is sign-extended and stored into the destination (leftmost) word operand.

(low byte DEST)  $\leftarrow$  (SRC)  
 if (SRC) < 80H then  
   (high byte DEST)  $\leftarrow$  0  
 else  
   (high byte DEST)  $\leftarrow$  0FFH  
 end\_if

**Assembly Language Format:**           DST     SRC  
 LDBSE   wreg,   baop

**Object Code Format:** [ 101111aa ][ baop ][ wreg ]

Bytes: 2 + BEA

States: 4 + CEA

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| -              | - | - | - | -  | -  |

**3.13.53. LDBZE — LOAD WORD WITH BYTE**

**Operation:** The value of the source (rightmost) byte operand is zero-extended and stored into the destination (leftmost) word operand.

(low byte DEST)  $\leftarrow$  (SRC)  
 (high byte DEST)  $\leftarrow$  0

**Assembly Language Format:**           DST     SRC  
 LDBZE   wreg,   baop

**Object Code Format:** [ 101011aa ][ baop ][ wreg ]

Bytes: 2 + BEA

States: 4 + CEA

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| -              | - | - | - | -  | -  |

**3.13.54. LJMP — LONG JUMP**

**Operation:** The distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The operand may be any address in the entire address space.

$$PC \leftarrow PC + \text{disp}$$

**Assembly Language Format:** LJMP cadd

**Object Code Format:** [ 11100111 ][ disp-low ][ disp-hi ]

Bytes: 3

States: 8

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| -              | - | - | - | -  | -  |

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| -              | - | - | - | -  | -  |

**3.13.55. MUL (Two Operands) — MULTIPLY INTEGERS**

**Operation:** The two INTEGER operands are multiplied using signed arithmetic and the 32-bit result is stored into the destination (leftmost) LONG-INTEGER operand. The sticky bit flag is undefined after the instruction is executed.

$$(\text{DEST}) \leftarrow (\text{DEST}) * (\text{SRC})$$

**Assembly Language Format:**           DST     SRC  
MUL lreg,     waop

**Object Code Format:** [ 11111110 ][ 011011aa ][ waop ][ lreg ]

Bytes: 3 + BEA

States: 29 + CEA

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| -              | - | - | - | -  | ?  |

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| -              | - | - | - | -  | -  |

**3.13.56. MUL (Three Operands) — MULTIPLY INTEGERS**

**Operation:** The second and third INTEGER operands are multiplied using signed arithmetic and the 32-bit result is stored into the destination (leftmost) INTEGER operand. The sticky bit flag is undefined after the instruction is executed.

$$(DEST) \leftarrow (SRC1) * (SRC2)$$

**Assembly Language Format:**

|     |       |       |      |
|-----|-------|-------|------|
|     | DST   | SRC1  | SRC2 |
| MUL | lreg, | wreg, | waop |

**Object Code Format:** [ 11111110 ][ 010011aa ][ waop ][ wreg ][ lreg ]

Bytes: 4 + BEA  
States: 30 + CEA

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| -              | - | - | - | -  | ?  |

**3.13.57. MULB (Two Operands) — MULTIPLY SHORT-INTEGERS**

**Operation:** The two SHORT-INTEGER operands are multiplied using signed arithmetic and the 16-bit result is stored into the destination (leftmost) INTEGER operand. The sticky bit flag is undefined after the instruction is executed.

$$(DEST) \leftarrow (DEST) * (SRC)$$

**Assembly Language Format:**

|      |       |      |
|------|-------|------|
|      | DST   | SRC  |
| MULB | wreg, | baop |

**Object Code Format:** [ 11111110 ][ 011111aa ][ baop ][ wreg ]

Bytes: 3 + BEA  
States: 21 + CEA

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| -              | - | - | - | -  | ?  |

### 3.13.58. MULB (Three Operands) — MULTIPLY SHORT-INTEGERS

**Operation:** The second and third SHORT-INTEGER operands are multiplied using signed arithmetic and the 16-bit result is stored into the destination (leftmost) INTEGER operand. The sticky bit flag is undefined after the instruction is executed.

$$(DEST) \leftarrow (SRC1) * (SRC2)$$

**Assembly Language Format:** `DST SRC1 SRC2`  
`MULB wreg, breg baop`

**Object Code Format:** `[ 11111110 ][ 010111aa ][ baop ][ breg ][ wreg ]`

Bytes: 4 + BEA

States: 22 + CEA

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| -              | - | - | - | -  | ?  |

### 3.13.59. MULU (Two Operands) — MULTIPLY WORDS

**Operation:** The two WORD operands are multiplied using unsigned arithmetic and the 32-bit result is stored into the destination (leftmost) DOUBLE-WORD operand. The sticky bit flag is undefined after the instruction is executed.

$$(DEST) \leftarrow (DEST) * (SRC)$$

**Assembly Language Format:** `DST SRC`  
`MULU lreg, waop`

**Object Code Format:** `[ 011011aa ][ waop ][ lreg ]`

Bytes: 2 + BEA

States: 25 + CEA

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| -              | - | - | - | -  | ?  |



**3.13.60. MULU (Three Operands) — MULTIPLY WORDS**

**Operation:** The second and third WORD operands are multiplied using unsigned arithmetic and the 32-bit result is stored into the destination (leftmost) DOUBLE-WORD operand. The sticky bit flag is undefined after the instruction is executed.

$$(DEST) \leftarrow (SRC1) * (SRC2)$$

**Assembly Language Format:** `DST SRC1 SRC2`  
`MULU lreg, wreg, waop`

**Object Code Format:** `[ 010011aa ][ waop ][ wreg ][ lreg ]`

Bytes: 3 + BEA

States: 26 + CEA

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| -              | - | - | - | -  | ?  |

**3.13.61. MULUB (Two Operands) — MULTIPLY BYTES**

**Operation:** The two BYTE operands are multiplied using unsigned arithmetic and the WORD result is stored into the destination (leftmost) operand. The sticky bit flag is undefined after the instruction is executed.

$$(DEST) \leftarrow (DEST) * (SRC)$$

**Assembly Language Format:** `DST SRC`  
`MULUB wreg, baop`

**Object Code Format:** `[ 011111aa ][ baop ][ wreg ]`

Bytes: 2 + BEA

States: 17 + CEA

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| -              | - | - | - | -  | ?  |

### 3.13.62. MULUB (Three Operands) — MULTIPLY BYTES

**Operation:** The second and third BYTE operands are multiplied using unsigned arithmetic and the WORD result is stored into the destination (leftmost) operand. The sticky bit flag is undefined after the instruction is executed.

$$(DEST) \leftarrow (SRC1) * (SRC2)$$

**Assembly Language Format:** DST SRC1 SRC2  
MULUB wreg, breg, baop

**Object Code Format:** [ 010111aa ][ baop ][ breg ][ wreg ]

Bytes: 3 + BEA  
States: 18 + CEA

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| -              | - | - | - | -  | ?  |

### 3.13.63. NEG — NEGATE INTEGER

**Operation:** The value of the INTEGER operand is negated.  
(DEST)  $\leftarrow$  -(DEST)

**Assembly Language Format:** NEG wreg

**Object Code Format:** [ 00000011 ][ wreg ]

Bytes: 2  
States: 4

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| ✓              | ✓ | ✓ | ✓ | ↑  | -  |

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| -              | - | - | - | -  | ?  |

**3.13.64. NEGB — NEGATE SHORT-INTEG**

**Operation:** The value of the SHORT-INTEG operand is negated.

$(DEST) \leftarrow -(DEST)$

**Assembly Language Format:** NEGB breg

**Object Code Format:** [ 00010011 ][ breg ]

Bytes: 2

States: 4

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| ✓              | ✓ | ✓ | ✓ | ↑  | —  |

**3.13.65. NOP — NO OPERATION**

**Operation:** Nothing is done. Control passes to the next sequential instruction.

**Assembly Language Format:** NOP

**Object Code Format:** [ 11111101 ]

Bytes: 1

States: 4

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| —              | — | — | — | —  | —  |

### 3.13.66. NORML — NORMALIZE LONG-INTEGER

**Operation:** The LONG-INTEGER operand is normalized; i.e., it is shifted to the left until its most significant bit is 1. If the most significant bit is still 0 after 31 shifts, the process stops and the zero flag is set. The number of shifts actually performed is stored in the second operand.

```
(COUNT) ← 0
do while (MSB(DEST) = 0) AND ((COUNT) < 31)
    (DEST) ← (DEST) * 2
    (COUNT) ← (COUNT) + 1
end _ while
```

**Assembly Language Format:** NORML lreg,breg

**Object Code Format:** [ 00001111 ][ breg ][ lreg ]

Bytes: 3

States: 8 + No. of shifts performed

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| ✓              | ? | 0 | — | —  | —  |

### 3.13.67. NOT — COMPLEMENT WORD

**Operation:** The value of the WORD operand is complemented: each 1 is replaced with a 0, and each 0 with a 1.

```
(DEST) ← NOT(DEST)
```

**Assembly Language Format:** NOT wreg

**Object Code Format:** [ 00000010 ][ wreg ]

Bytes: 2

States: 4

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| ✓              | ✓ | 0 | 0 | —  | —  |



**3.13.68. NOTB — COMPLEMENT BYTE**

**Operation:** The value of the BYTE operand is complemented: each 1 is replaced with a 0, and each 0 with a 1.

(DEST) ← NOT (DEST)

**Assembly Language Format:** NOTB breg

**Object Code Format:** [ 00010010 ][ breg ]

Bytes: 2

States: 4

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| ✓              | ✓ | 0 | 0 | -  | -  |

**3.13.69. OR — LOGICAL OR WORDS**

**Operation:** The source (rightmost) WORD is ORed with the destination (leftmost) WORD operand. Each bit is set to 1 if the corresponding bit in either the source operand or the destination operand is 1. The result replaces the original destination operand.

(DEST) ← (DEST) OR (SRC)

**Assembly Language Format:** DST SRC  
OR wreg, waop

**Object Code Format:** [ 100000aa ][ waop ][ wreg ]

Bytes: 2 + BEA

States: 4 + CEA

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| ✓              | ✓ | 0 | 0 | -  | -  |

**3.13.70. ORB — LOGICAL OR BYTES**

**Operation:** The source (rightmost) BYTE operand is ORed with the destination (leftmost) BYTE operand. Each bit is set to 1 if the corresponding bit in either the source operand or the destination operand was 1. The result replaces the original destination operand.

$(\text{DEST}) \leftarrow (\text{DEST}) \text{ OR } (\text{SRC})$

**Assembly Language Format:** ORB breg,baop

**Object Code Format:** [ 100100aa ][ baop ][ breg ]

Bytes: 2+BEA

States: 4+CEA

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| ✓              | ✓ | 0 | 0 | —  | —  |

**3.13.71. POP — POP WORD**

**Operation:** The word on top of the stack is popped and placed at the destination operand.

$(\text{DEST}) \leftarrow (\text{SP})$

$\text{SP} \leftarrow \text{SP} + 2$

**Assembly Language Format:** POP waop

**Object Code Format:** [ 110011aa ][ waop ]

Bytes:

1+BEA

States: Onchip Stack: 12+CEA

Offchip Stack: 14+CEA

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| —              | — | — | — | —  | —  |

**3.13.72. POPF — POP FLAGS**

**Operation:** The word on top of the stack is popped and placed in the PSW. Interrupt calls cannot occur immediately following this instruction.

(PSW)  $\leftarrow$  (SP)

SP  $\leftarrow$  SP + 2

**Assembly Language Format:** POPF

**Object Code Format:** [ 11110011 ]

Bytes: 1

States: Onchip Stack: 9

Offchip Stack: 13

Flags Affected

| Z | N | C | V | VT | ST |
|---|---|---|---|----|----|
| ✓ | ✓ | ✓ | ✓ | ✓  | ✓  |

**3.13.73. PUSH — PUSH WORD**

**Operation:** The specified operand is pushed onto the stack.

SP  $\leftarrow$  SP - 2

(SP)  $\leftarrow$  (DEST)

**Assembly Language Format:** PUSH waop

**Object Code Format:** [ 110010aa ][ waop ]

Bytes: 1 + BEA

States: Onchip Stack: 8 + CEA

Offchip Stack: 12 + CEA

Flags Affected

| Z | N | C | V | VT | ST |
|---|---|---|---|----|----|
| - | - | - | - | -  | -  |

### 3.13.74. PUSHF — PUSH FLAGS

**Operation:** The PSW is pushed on top of the stack, and then set to all zeroes. This implies that all interrupts are disabled. Interrupt-calls cannot occur immediately following this instruction.

$SP \leftarrow SP - 2$

$(SP) \leftarrow PSW$

$PSW \leftarrow 0$

**Assembly Language Format:** PUSHF

**Object Code Format:** [ 11110010 ]

Bytes: 1

States: Onchip Stack: 8

Offchip Stack: 12

Flags Affected

| Z | N | C | V | VT | ST |
|---|---|---|---|----|----|
| 0 | 0 | 0 | 0 | 0  | 0  |

### 3.13.75. RET — RETURN FROM SUBROUTINE

**Operation:** The PC is popped off the top of the stack.

$PC \leftarrow (SP)$

$SP \leftarrow SP + 2$

**Assembly Language Format:** RET

**Object Code Format:** [ 11110000 ]

Bytes: 1

States: Onchip Stack: 12

Offchip Stack: 16

Flags Affected

| Z | N | C | V | VT | ST |
|---|---|---|---|----|----|
| - | - | - | - | -  | -  |



**3.13.76. RST — RESET SYSTEM**

**Operation:** The PSW is initialized to zero, and the PC is initialized to 2080H. The I/O registers are set to their initial value (see section 2.15.2, "Reset Status"). Executing this instruction will cause a pulse to appear on the reset pin of the 8096.

PSW  $\leftarrow$  0  
PC  $\leftarrow$  2080H

**Assembly Language Format:** RST

**Object Code Format:** [ 11111111 ]

Bytes: 1  
States: 16

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| 0              | 0 | 0 | 0 | 0  | 0  |

**3.13.77. SCALL — SHORT CALL**

**Operation:** The contents of the program counter (the return address) is pushed onto the stack. Then the distance from the end of this instruction to the target label is added to the program counter, effecting the call. The offset from the end of this instruction to the target label must be in the range of -1024 to +1023 inclusive.

SP  $\leftarrow$  SP - 2  
(SP)  $\leftarrow$  PC  
PC  $\leftarrow$  PC + disp (sign-extended to 16 bits)

**Assembly Language Format:** SCALL cadd

**Object Code Format:** [ 00101xxx ] [ disp-low ]

where xxx holds the three high-order bits of displacement.

Bytes: 2  
States: Onchip Stack: 13  
Offchip Stack: 16

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| -              | - | - | - | -  | -  |

**3.13.78. SETC — SET CARRY FLAG****Operation:** The carry flag is set. $C \leftarrow 1$ **Assembly Language Format:** SETC**Object Code Format:** [ 11111001 ]

Bytes: 1

States: 4

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| -              | - | 1 | - | -  | -  |

**3.13.79. SHL — SHIFT WORD LEFT**

**Operation:** The destination (leftmost) word operand is shifted left as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register. Details on indirect shifts can be found in section 3.13. The right bits of the result are filled with zeroes. The last bit shifted out is saved in the carry flag.

```

Temp ← (COUNT)
do while Temp <> 0
  C ← High order bit of (DEST)
  (DEST) ← (DEST)*2
  Temp ← Temp - 1
end _ while

```

**Assembly Language Format:** SHL wreg,#count

or

SHL wreg,breg

**Object Code Format:** [ 00001001 ] [ cnt/breg ] [ wreg ]

Bytes: 3

States: 7 + No. of shifts performed

note: 0 place shifts take 8 states.

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| ✓              | ? | ✓ | ✓ | ↑  | -  |

**3.13.80. SHLB — SHIFT BYTE LEFT**

**Operation:** The destination (leftmost) byte operand is shifted left as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register. Details on indirect shifts can be found in section 3.13. The right bits of the result are filled with zeroes. The last bit shifted out is saved in the carry flag.

```
Temp ← (COUNT)
do while Temp <> 0
  C ← High order bit of (DEST)
  (DEST) ← (DEST)*2
  Temp ← Temp - 1
end _ while
```

**Assembly Language Format:** SHLB breg,#count  
or

SHLB breg,breg

**Object Code Format:** [ 00011001 ][ cnt/breg ][ breg ]

Bytes: 3

States: 7 + No. of shifts performed

note: 0 place shifts take 8 states.

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| ✓              | ? | ✓ | ✓ | ↑  | —  |

**Operation:** The destination (leftmost) double-word operand is shifted left as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register. Details on indirect shifts can be found in section 3.13. The right bits of the result are filled with zeroes. The last bit shifted out is saved in the carry flag.

```
Temp ← (COUNT)
do while Temp <> 0
    C ← High order bit of (DEST)
    (DEST) ← (DEST)*2
    Temp ← Temp - 1
end _ while
```

**Assembly Language Format:** SHLL lreg,#count  
or  
SHLL lreg,breg

**Object Code Format:** [ 00001101 ][ cnt/breg ][ lreg ]

Bytes: 3

States: 7 + No. of shifts performed

note: 0 place shifts take 8 states.

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| ✓              | ? | ✓ | ✓ | ↑  | -  |



**Operation:** The destination (leftmost) word operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register. Details on indirect shifts can be found in section 3.13. The left bits of the result are filled with zeroes. The last bit shifted out is saved to the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp <> 0
    C ← Low order bit of (DEST)
    (DEST) ← (DEST) / 2 where / is unsigned division
    Temp ← Temp - 1
end _ while
```

**Assembly Language Format:** SHR wreg,#count

or  
SHR wreg,breg

**Object Code Format:** [ 00001000 ][ cnt/breg ][ wreg ]

Bytes: 3

States: 7 + No. of shifts performed

note: 0 place shifts take 8 states.

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| ✓              | 0 | ✓ | 0 | -  | ✓  |

**3.13.83. SHRA — ARITHMETIC RIGHT SHIFT WORD**

**Operation:** The destination (leftmost) word operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register. Details on indirect shifts can be found in section 3.13. If the original high order bit value was 0, zeroes are shifted in. If the value was 1, ones are shifted in. The last bit shifted out is saved in the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp <> 0
    C ← Low order bit of (DEST)
    (DEST) ← (DEST) / 2 where / is signed division
    Temp ← Temp - 1
end _ while
```

**Assembly Language Format:** SHRA wreg,#count  
or  
SHRA wreg,breg

**Object Code Format:** [ 00001010 ][ cnt/breg ][ wreg ]

Bytes: 3

States: 7 + No. of shifts performed

note: 0 place shifts take 8 states.

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| ✓              | ✓ | ✓ | 0 | -  | ✓  |

**3.13.84. SHRAB — ARITHMETIC RIGHT SHIFT BYTE**

**Operation:** The destination (leftmost) byte operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register. Details on indirect shifts can be found in section 3.13. If the original high order bit value was 0, zeroes are shifted in. If that value was 1, ones are shifted in. The last bit shifted out is saved in the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp <> 0
  C, = Low order bit of (DEST)
  (DEST) ← (DEST) / 2 where / is signed division
  Temp ← Temp - 1
end _ while
```

**Assembly Language Format:** SHRAB breg, #count  
or  
SHRAB breg, breg

**Object Code Format:** [ 00011010 ] [ cnt/breg ] [ breg ]

Bytes: 3

States: 7 + No. of shifts performed

note: 0 place shifts take 8 states.

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| ✓              | ✓ | ✓ | 0 | -  | ✓  |

**Operation:** The destination (leftmost) double-word operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register. Details on indirect shifts can be found in section 3.13. If the original high order bit value was 0, zeroes are shifted in. If the value was 1, ones are shifted in. The sticky bit is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

Temp ← (COUNT)

do while Temp < > 0

C ← Low order bit of (DEST)

(DEST) ← (DEST) / 2 where / is signed division

Temp ← Temp - 1

end \_ while

**Assembly Language Format:** SHRAL reg, #count

or

SHRAL reg, breg

**Object Code Format:** [ 00001110 ] [ cnt/breg ] [ :reg ]

Bytes: 3

States: 7 + No. of shifts performed

note: 0 place shifts take 8 states.

Flags Affected

| Z | N | C | V | VT | ST |
|---|---|---|---|----|----|
| √ | √ | √ | 0 | -  | √  |



### 3.13.86. SHRB — LOGICAL RIGHT SHIFT BYTE

**Operation:** The destination (leftmost) byte operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register. Details on indirect shifts can be found in section 3.13. The left bits of the result are filled with zeroes. The last bit shifted out is saved in the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp <> 0
  C ← Low order bit of (DEST)
  (DEST) ← (DEST) / 2 where / is unsigned division
  Temp ← Temp - 1
end _ while
```

**Assembly Language Format:** SHRB breg, #count  
or  
SHRB breg, breg

**Object Code Format:** [ 00011000 ] [ cnt/breg ] [ breg ]

Bytes: 3

States: 7 + No. of shifts performed

note: 0 place shifts take 8 states.

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| ✓              | 0 | ✓ | 0 | -  | ✓  |

### 3.13.87. SHRL — LOGICAL RIGHT SHIFT DOUBLE-WORD

**Operation:** The destination (leftmost) double-word operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 (0FH) inclusive, or as the content of any register. Details on indirect shifts can be found in section 3.13. The left bits of the result are filled with zeroes. The last bit shifted out is saved in the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp < > 0
  C ← Low order bit of (DEST)
  (DEST) ← (DEST) / 2 where / is unsigned division
  Temp ← Temp - 1
end _ while
```

**Assembly Language Format:** SHRL lreg, #count

or

SHRL lreg, breg

**Object Code Format:** [ [ 00001100 ] [ cnt/breg ] [ .lreg ]

Bytes: 3

States: 7 + No. of shifts performed

note: 0 place shifts take 8 states.

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| ✓              | 0 | ✓ | 0 | -  | ✓  |

**3.13.88. SJMP — SHORT JUMP**

**Operation:** The distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -1024 to +1023 inclusive.

$PC \leftarrow PC + \text{disp (sign-extended to 16 bits)}$

**Assembly Language Format:** SJMP cadd

**Object Code Format:** [ 00100xxx ][ disp-low ]

where xxx holds the three high order bits of the displacement.

Bytes: 2

States: 8

Flags Affected

| Z | N | C | V | VT | ST |
|---|---|---|---|----|----|
| - | - | - | - | -  | -  |

**3.13.89. SKIP — TWO BYTE NO-OPERATION**

**Operation:** Nothing is done. This is actually a two-byte NOP where the second byte can be any value, and is simply ignored. Control passes to the next sequential instruction.

**Assembly Language Format:** SKIP breg

**Object Code Format:** [ 00000000 ][ breg ]

Bytes: 2

States: 4

Flags Affected

| Z | N | C | V | VT | ST |
|---|---|---|---|----|----|
| - | - | - | - | -  | -  |

### 3.13.90. ST — STORE WORD

**Operation:** The value of the leftmost word operand is stored into the rightmost operand.  
 $(DEST) \leftarrow (SRC)$

**Assembly Language Format:** SRC DST  
 ST wreg, waop

**Object Code Format:** [ 110000aa ] [ waop ] [ wreg ]

Bytes: 2 + BEA

States: 4 + CEA

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| -              | - | - | - | -  | -  |

### 3.13.91. STB — STORE BYTE

**Operation:** The value of the leftmost byte operand is stored into the rightmost operand.

$(DEST) \leftarrow (SRC)$

**Assembly Language Format:** SRC DST  
 STB breg, baop

**Object Code Format:** [ 110001aa ] [ baop ] [ breg ]

Bytes: 2 + BEA

States: 4 + CEA

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| -              | - | - | - | -  | -  |



### 3.13.92. SUB (Two Operands) — SUBTRACT WORDS

**Operation:** The source (rightmost) word operand is subtracted from the destination (leftmost) word operand, and the result is stored in the destination. The carry flag is set as complement of borrow.

$$(DEST) \leftarrow (DEST) - (SRC)$$

**Assembly Language Format:**      DST      SRC  
SUB    wreg,    waop

**Object Code Format:** [ 011010aa ][ waop ][ wreg ]

Bytes: 2+BEA  
States: 4+CEA

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| ✓              | ✓ | ✓ | ✓ | ↑  | —  |

### 3.13.93. SUB (Three Operands) — SUBTRACT WORDS

**Operation:** The source (rightmost) word operand is subtracted from the second word operand, and the result is stored in the destination (the leftmost operand). The carry flag is set as complement of borrow.

$$(DEST) \leftarrow (SRC1) - (SRC2)$$

**Assembly Language Format:**      DST      SRC1      SRC2  
SUB    wreg,    wreg,    waop

**Object Code Format:** [ 010010aa ][ waop ][ Swreg ][ Dwreg ]

Bytes: 3+BEA  
States: 5+CEA

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| ✓              | ✓ | ✓ | ✓ | ↑  | —  |

**3.13.94. SUBB (Two Operands) — SUBTRACT BYTES**

**Operation:** The source (rightmost) byte is subtracted from the destination (leftmost) byte operand, and the result is stored in the destination. The carry flag is set as complement of borrow.

$$(DEST) \leftarrow (DEST) - (SRC)$$

**Assembly Language Format:**            DST    SRC  
SUBB   breg,   baop

**Object Code Format:** [ 011110aa ][ baop ][ breg ]

Bytes: 2 + BEA

States: 4 + CEA

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| ✓              | ✓ | ✓ | ✓ | ↑  | -  |

**3.13.95. SUBB (Three Operands) — SUBTRACT BYTES**

**Operation:** The source (rightmost) byte operand is subtracted from the second byte operand, and the result is stored in the destination (the leftmost operand). The carry flag is set as complement of borrow.

$$(DEST) \leftarrow (SRC1) - (SRC2)$$

**Assembly Language Format:**            DST    SRC1    SRC2  
SUBB   breg,   Sbreg   baop

**Object Code Format:** [ 010110aa ][ baop ][ Sbreg ][ Dbreg ]

Bytes: 3 + BEA

States: 5 + CEA

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| ✓              | ✓ | ✓ | ✓ | ↑  | -  |

**3.13.96. SUBC — SUBTRACT WORDS WITH BORROW**

**Operation:** The source (rightmost) word operand is subtracted from the destination (leftmost) word operand. If the carry flag was clear, 1 is subtracted from the above result. The result replaces the original destination operand. The carry flag is set as complement of borrow.

$$(DEST) \leftarrow (DEST) - (SRC) - (1-C)$$

**Assembly Language Format:**

|      |       |      |
|------|-------|------|
|      | DST   | SRC  |
| SUBC | wreg, | waop |

**Object Code Format:** [ 101010aa ][ waop ][ wreg ]

Bytes: 2+BEA

States: 4+CEA

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| ↓              | ✓ | ✓ | ✓ | ↑  | —  |

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| —              | — | — | — | —  | —  |

**3.13.97. SUBCB — SUBTRACT BYTES WITH BORROW**

**Operation:** The source (rightmost) byte operand is subtracted from the destination (leftmost) byte operand. If the carry flag was clear, 1 is subtracted from the above result. The result replaces the original destination operand. The carry flag is set as complement of borrow.

$$(DEST) \leftarrow (DEST) - (SRC) - (1-C)$$

**Assembly Language Format:**

|       |       |      |
|-------|-------|------|
|       | DST   | SRC  |
| SUBCB | breg, | baop |

**Object Code Format:** [ 101110aa ][ baop ][ breg ]

Bytes: 2+BEA

States: 4+CEA

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| ↓              | ✓ | ✓ | ✓ | ↑  | —  |

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| —              | — | — | — | —  | —  |

**3.13.98. TRAP — SOFTWARE TRAP**

**Operation:** This instruction causes an interrupt-call which is vectored through location 2010H. The operation of this instruction is not effected by the state of the interrupt enable flag in the PSW (I). Interrupt-calls cannot occur immediately following this instruction. This instruction is intended for use by Intel provided development tools. These tools will not support user-application of this instruction.

$SP \leftarrow SP - 2$

$(SP) \leftarrow PC$

$PC \leftarrow (2010H)$

**Assembly Language Format:** This instruction is not supported by revision 1.0 of the 8096 assembly language.

**Object Code Format:** [ 11110111 ]

Bytes: 1

States: Onchip Stack: 21

Offchip Stack: 24

Flags Affected

| Z | N | C | V | VT | ST |
|---|---|---|---|----|----|
| - | - | - | - | -  | -  |

**3.13.99. XOR — LOGICAL EXCLUSIVE-OR WORDS**

**Operation:** The source (rightmost) word operand is XORed with the destination (leftmost) word operand. Each bit is set to 1 if the corresponding bit in either the source operand or the destination operand was 1, but not both. The result replaces the original destination operand.

$(DEST) \leftarrow (DEST) \text{ XOR } (SRC)$

**Assembly Language Format:**           DST   SRC

XOR   wreg,   waop

**Object Code Format:** [ 100001aa ][ waop ][ wreg ]

Bytes: 2+BEA

States: 4+CEA

Flags Affected

| Z | N | C | V | VT | ST |
|---|---|---|---|----|----|
| ✓ | ✓ | 0 | 0 | -  | -  |



---

**3.13.100. XORB — LOGICAL EXCLUSIVE-OR BYTES**

---

**Operation:** The source (rightmost) byte operand is XORed with the destination (leftmost) byte operand. Each bit is set to 1 if the corresponding bit in either the source operand or the destination operand was 1, but not both. The result replaces the original destination operand.

$$(\text{DEST}) \leftarrow (\text{DEST}) \text{ XOR } (\text{SRC})$$

**Assembly Language Format:**           DST     SRC  
XORB   breg,   baop

**Object Code Format:** [ 100101aa ][ baop ][ breg ]

Bytes: 2 + BEA

States: 4 + CEA

| Flags Affected |   |   |   |    |    |
|----------------|---|---|---|----|----|
| Z              | N | C | V | VT | ST |
| ✓              | ✓ | 0 | 0 | —  | —  |



---

**MCS®-96 Hardware  
Design Information**

---

**4**





# CHAPTER 4

## MCS®-96 HARDWARE DESIGN INFORMATION

### 4.0. HARDWARE INTERFACING OVERVIEW

This section of the manual is devoted to the hardware engineer. All of the information you need to connect the correct pin to the correct external circuit is provided. Many of the special function pins have different characteristics which are under software control, therefore, it is necessary to define the system completely before the hardware is wired-up.

Frequently within this section a specification for a current, voltage, or time period is referred to; the values provided are to be used as an approximation only. The exact specification can be found in the latest data sheet for the particular part and temperature range that is being used.

### 4.1. REQUIRED HARDWARE CONNECTIONS

Although the 8096 is a single-chip microcontroller, it still requires several external connections to make it work. Power must be applied, a clock source provided, and some form of reset circuitry must be present. We will look at each of these areas of circuitry separately. Figure 4-5 shows the connections that are needed for a single-chip system.

#### 4.1.1. Power Supply Information

Power for 8096 flows through 6 pins; one VCC pin, two VSS pins, one VREF (analog VCC), one ANGND (Analog VSS), and one VPD (V Power Down) pin. All six of these pins must be connected to the 8096 for normal operation. The VCC pin, VREF pin and VPD pin should be tied to 5 volts. When the analog to digital converter is being used it may be desirable to connect the VREF

pin to a separate power supply, or at least a separate power supply line.

The two VSS pins should be connected together with as short a lead as possible to avoid problems due to voltage drops across the wiring. There should be no measurable voltage difference between VSS1 and VSS2. The 2 VSS pins and the ANGND pin should all be nominally at 0 volts. The maximum current drain of the 8096 is around 200mA, with all lines unloaded.

When the analog converter is being used, clean, stable power must be provided to the analog section of the chip to assure highest accuracy. To achieve this, it may be desirable to separate the analog power supply from the digital power supply. The VREF pin supplies 5 volts to the analog circuitry and the ANGND pin is the ground for this section of the chip. More information on the analog power supply is in section 4.3.1.

#### 4.1.2. Other Needed Connections

Several of the pins on the 8096 are used to configure the mode of operation. The 8096BH has additional features which make use of some of these pins. See the 8096BH data sheet in Chapter 5 for more information. In normal operation the following pins should be tied directly to the indicated power supply.

| PIN                      | POWER SUPPLY   |
|--------------------------|--|
| NMI                      | VCC  |
| $\overline{\text{TEST}}$ | VCC  |
| $\overline{\text{EA}}$   | VCC (to allow internal execution)<br>VSS (to force external execution) |

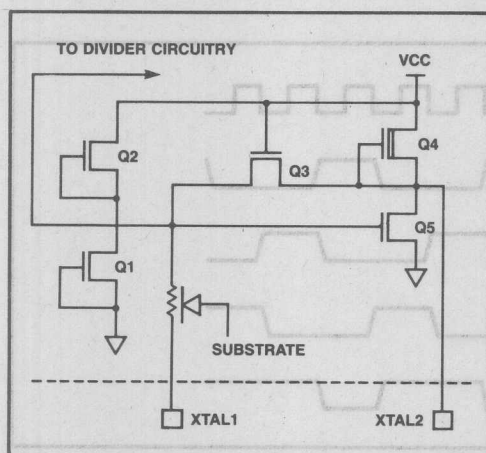


Figure 4-1. 8096 Oscillator Circuit

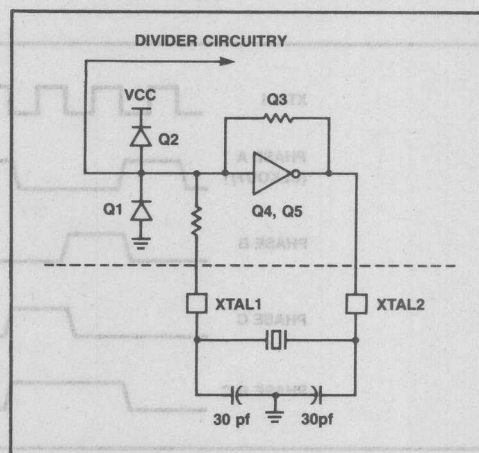


Figure 4-2. Crystal Oscillator Circuit

Although the EA pin has an internal pulldown, it is best to tie this pin to the desired level if it is not left completely disconnected. This will prevent induced noise from disturbing the system.

#### 4.1.3. Oscillator Information

The 8096 requires a clock source to operate. This clock can be provided to the chip through the XTAL1 input or the on-chip oscillator can be used. The frequency of operation is from 6.0 MHz to 12 MHz.

The on-chip circuitry for the 8096 oscillator is a single stage linear inverter as shown in Figure 4-1. It is intended for use as a crystal-controlled, positive reactance oscillator with external connections as shown in Figure 4-2. In this application, the crystal is being operated in its fundamental

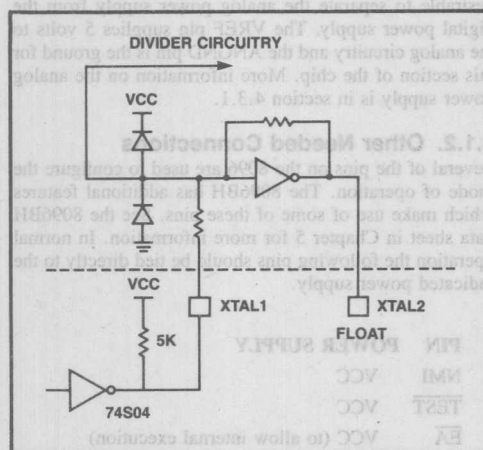


Figure 4-3. External Clock Drive

response mode as an inductive reactance in parallel resonance with capacitance external to the crystal.

The crystal specifications and capacitance values (C1 and C2 in Figure 4-2) are not critical. 30 pF can be used in these positions at any frequency with good quality crystals. For 0.5% frequency accuracy, the crystal frequency can be specified at series resonance or for parallel resonance with any load capacitance. (In other words, for that degree of frequency accuracy, the load capacitance simply doesn't matter.) For 0.05% frequency accuracy the crystal frequency should be specified for parallel resonance with 25 pF load capacitance, if C1 and C2 are 30 pF.

A more in-depth discussion of crystal specifications and the selection of values for C1 and C2 can be found in the Intel Application Note, AP-155, "Oscillators for Microcontrollers."

To drive the 8096 with an external clock source, apply the external clock signal to XTAL1 and let XTAL2 float. An example of this circuit is shown in Figure 4-3. The required voltage levels on XTAL1 are specified in the data sheet. The signal on XTAL1 must be clean with good solid levels. It is important that the minimum high and low times are met.

There is no specification on rise and fall times, but they should be reasonably fast (on the order of 30 nanoseconds) to avoid having the XTAL1 pin in the transition range for long periods of time. The longer the signal is in the transition region, the higher the probability that an external noise glitch could be seen by the clock generator circuitry. Noise glitches on the 8096 internal clock lines will cause unreliable operation.

The clock generator provides a 3 phase clock output from the XTAL1 pin input. Figure 4-4 shows the waveforms of the major internal timing signals.

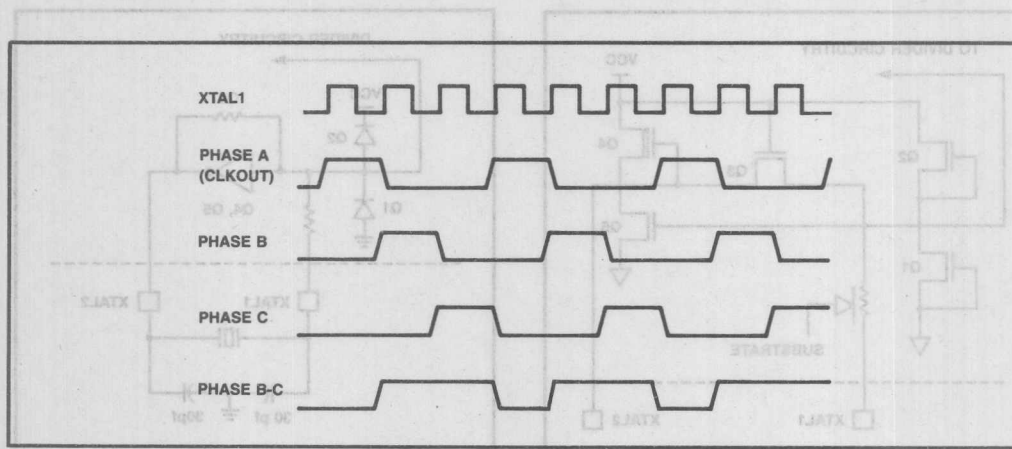


Figure 4-4. Internal Timings

#### 4.1.4. Reset Information

There are minor enhancements to the reset sequence for the 8096BH. See the data sheet in Chapter 5. In order for the 8096 to function properly it must be reset. This is done by holding the reset pin low for at least 2 state times after the power supply is within tolerance, the oscillator has stabilized, and the back-bias generator has stabilized. Typically, the back-bias generator requires one millisecond to stabilize.

There are several ways of doing this, the simplest being just to connect a capacitor from the reset pin to ground. The capacitor should be on the order of 1 to 2 microfarads for every millisecond of reset time required. This method will only work if the rise time of VCC is fast and the total reset time is less than around 50 milliseconds. It also may not work if the reset pin is to be used to reset other parts on the board. An 8096 with the minimum required connections is shown in Figure 4-5.

The 8096  $\overline{\text{RESET}}$  pin can be used to allow other chips on the board to make use of the watchdog timer or the RST instruction. When this is done the reset hardware should be a one-shot with an open-collector output. The reset

pulse going to the other parts may have to be buffered and lengthened with a one-shot, since the  $\overline{\text{RESET}}$  low duration is only two state times. If this is done, it is possible that the 8096 will be reset and start running before the other parts on the board are out of reset. The software must account for this possible problem.

A capacitor directly connected to  $\overline{\text{RESET}}$  cannot be used to reset the part if the pin is to be used as an output. If a large capacitor is used, the pin will pull down more slowly than normal. It will continue to pull down until the 8096 is reset. It could fall so slowly that it never goes below the internal switch point of the reset signal (1 to 1.5 volts), a voltage which may be above the guaranteed switch point of external circuitry connected to the pin. A circuit example is shown in Figure 4-6.

#### 4.1.5. Sync Mode

If  $\overline{\text{RESET}}$  is brought high at the same time as or just after the rising edge of XTAL1, the part will start executing the 10 state time RST instruction exactly  $6\frac{1}{2}$  XTAL1 cycles later. This feature can be used to synchronize several MCS-96 devices. A diagram of a typical connection is shown in Figure 4-7. It should be noted that parts that

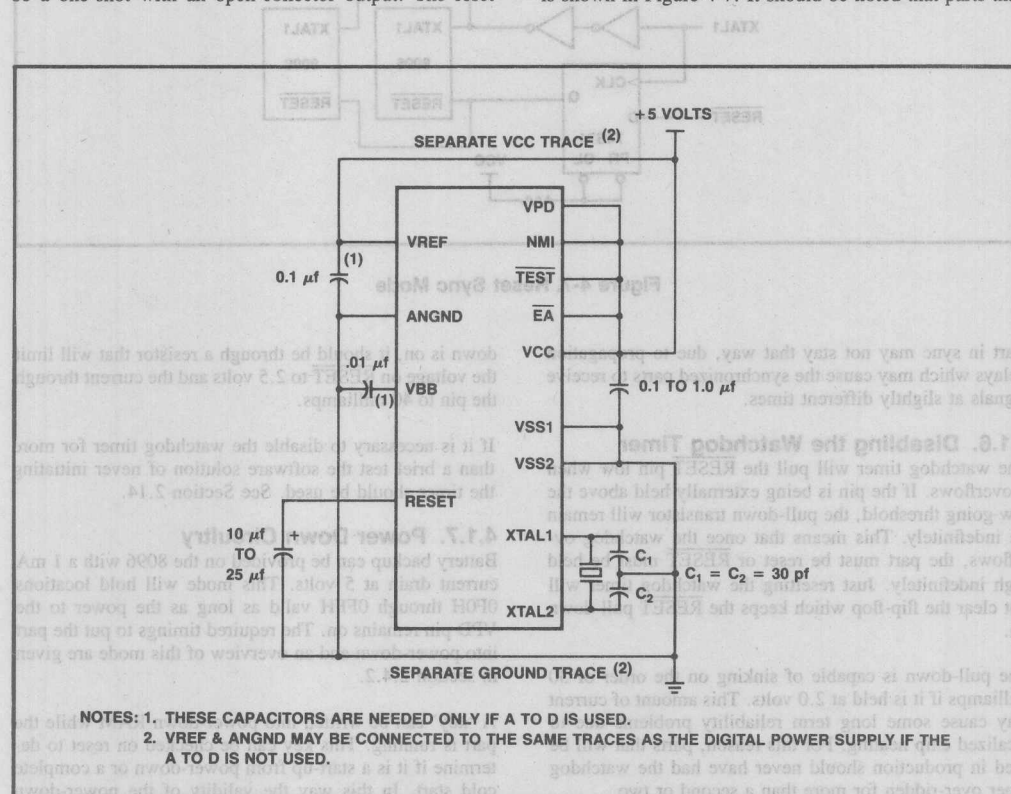


Figure 4-5. Minimum Hardware Connections

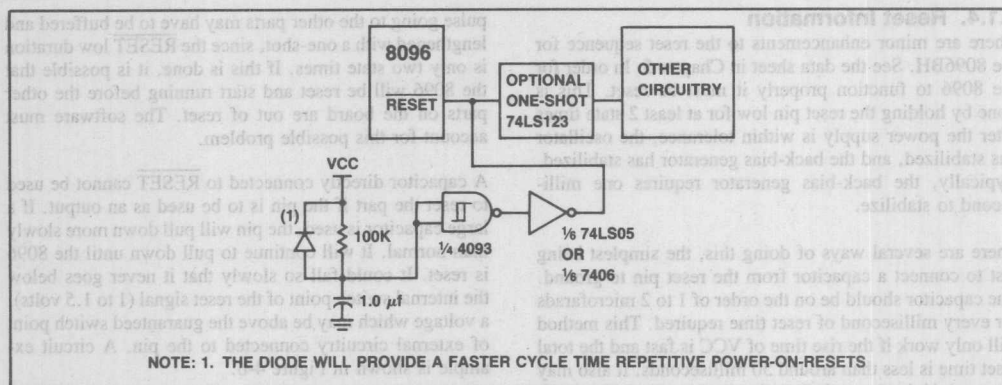


Figure 4-6. Multiple Chip Reset Circuit

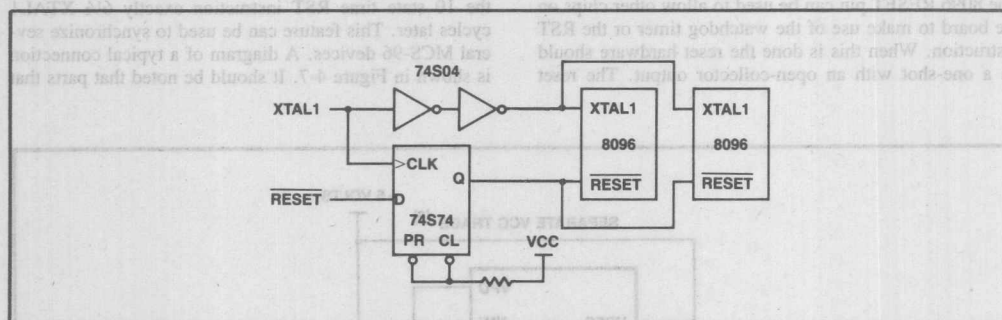


Figure 4-7. Reset Sync Mode

start in sync may not stay that way, due to propagation delays which may cause the synchronized parts to receive signals at slightly different times.

#### 4.1.6. Disabling the Watchdog Timer

The watchdog timer will pull the **RESET** pin low when it overflows. If the pin is being externally held above the low going threshold, the pull-down transistor will remain on indefinitely. This means that once the watchdog overflows, the part must be reset or **RESET** must be held high indefinitely. Just resetting the watchdog timer will not clear the flip-flop which keeps the **RESET** pull-down on.

The pull-down is capable of sinking on the order of 30 milliamps if it is held at 2.0 volts. This amount of current may cause some long term reliability problems due to localized chip heating. For this reason, parts that will be used in production should never have had the watchdog timer over-ridden for more than a second or two.

Whenever the reset pin is being pulled high while the pull-

down is on, it should be through a resistor that will limit the voltage on **RESET** to 2.5 volts and the current through the pin to 40 milliamps.

If it is necessary to disable the watchdog timer for more than a brief test the software solution of never initiating the timer should be used. See Section 2.14.

#### 4.1.7. Power Down Circuitry

Battery backup can be provided on the 8096 with a 1 mA current drain at 5 volts. This mode will hold locations 0FOH through 0FFH valid as long as the power to the VPD pin remains on. The required timings to put the part into power-down and an overview of this mode are given in section 2.4.2.

A 'key' can be written into power-down RAM while the part is running. This key can be checked on reset to determine if it is a start-up from power-down or a complete cold start. In this way the validity of the power-down RAM can be verified. The length of this key determines the probability that this procedure will work, however,



there is always a statistical chance that the RAM will power up with a replica of the key.

Under most circumstances, the power-fail indicator which is used to initiate a power-down condition must come from the unfiltered, unregulated section of the power supply. The power supply must have sufficient storage capacity to operate the 8096 until it has completed its reset operation.

## 4.2. DRIVE AND INTERFACE LEVELS

There are 5 types of I/O lines on the 8096. Of these, 2 are inputs and 3 are outputs. All of the pins of the same type have the same current/voltage characteristics. Some of the control input pins, such as XTAL1 and RESET, may have slightly different characteristics. These pins are discussed in section 4.1.1. While discussing the characteristics of the I/O pins some approximate current or voltage specifications will be given. The exact specifications are available in the latest version of the 8096 Data Sheet.

### 4.2.1. Quasi-Bidirectional Ports

The quasi-bidirectional port is both an input and an output port. It has three states, low impedance current sink, low impedance current source, and high impedance current source. As a low impedance current sink, the pin has a specification of sinking up to around .4 milliamps, while staying below 0.45 volts. The pin is placed in this condition by writing a '0' to the SFR (Special Function Register) controlling the pin.

When a '1' is written to the SFR location controlling the pin, a low impedance current source is turned on for one state time, then it is turned off and the depletion pull-up holds the line at a logical '1' state. The low-impedance pull-up is used to shorten the rise time of the pin, and has current source capability on the order of 100 times that of the depletion pull-up. The configuration of a quasi-bidirectional port pin is shown in Figure 4-8.

While the depletion mode pull-up is the only device on, the pin may be used as an input with a leakage of around 100 microamps from 0.45 volts to VCC. It is ideal for

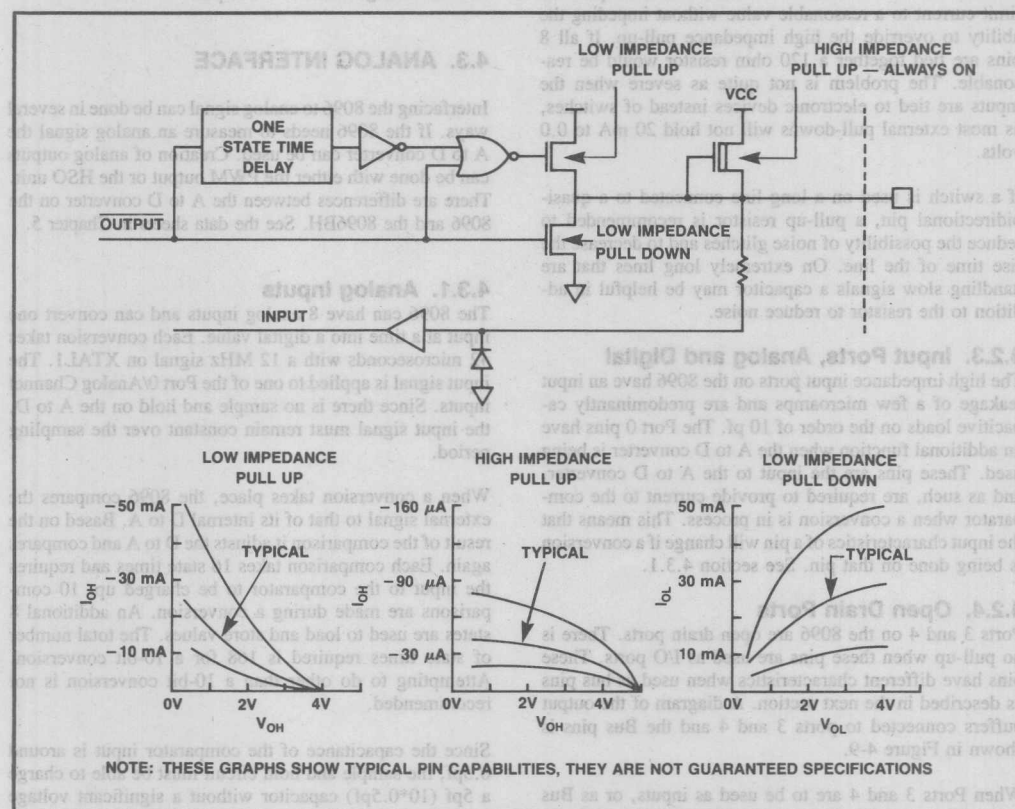


Figure 4-8. Quasi-Bidirectional Port

use with TTL or CMOS chips and may even be used directly with switches, however if the switch option is used certain precautions should be taken. It is important to note that any time the pin is read, the value returned will be the value on the pin, not the value placed in the control register. This could prevent logical operations on these pins while they are being used as inputs.

#### 4.2.2. Quasi-Bidirectional Hardware Connections

When using the quasi-bidirectional ports as inputs tied to switches, series resistors may be needed if the ports will be written to internally after the part is initialized. The amount of current sourced to ground by each pin is typically 20 mA or more. Therefore, if all 8 pins are tied to ground, 160 mA will be sourced. This is equivalent to instantaneously doubling the power used by the chip and may cause noise in some applications.

This potential problem can be solved in hardware or software. In software, never write a zero to a pin being used as an input.

In hardware, a 1K resistor in series with each pin will limit current to a reasonable value without impeding the ability to override the high impedance pull-up. If all 8 pins are tied together a 120 ohm resistor would be reasonable. The problem is not quite as severe when the inputs are tied to electronic devices instead of switches, as most external pull-downs will not hold 20 mA to 0.0 volts.

If a switch is used on a long line connected to a quasi-bidirectional pin, a pull-up resistor is recommended to reduce the possibility of noise glitches and to decrease the rise time of the line. On extremely long lines that are handling slow signals a capacitor may be helpful in addition to the resistor to reduce noise.

#### 4.2.3. Input Ports, Analog and Digital

The high impedance input ports on the 8096 have an input leakage of a few microamps and are predominantly capacitive loads on the order of 10 pf. The Port 0 pins have an additional function when the A to D converter is being used. These pins are the input to the A to D converter, and as such, are required to provide current to the comparator when a conversion is in process. This means that the input characteristics of a pin will change if a conversion is being done on that pin. See section 4.3.1.

#### 4.2.4. Open Drain Ports

Ports 3 and 4 on the 8096 are open drain ports. There is no pull-up when these pins are used as I/O ports. These pins have different characteristics when used as bus pins as described in the next section. A diagram of the output buffers connected to ports 3 and 4 and the Bus pins is shown in Figure 4-9.

When Ports 3 and 4 are to be used as inputs, or as Bus pins, they must first be written with a '1', this will put the ports in a high impedance mode. When they are used

as outputs, a pull-up resistor must be used externally. The sink capability of these pins is on the order of 0.4 milliamps so the total pull-up current to the pin must be less than this. A 15k pull-up resistor will source a maximum of 0.33 milliamps, so it would be a reasonable value to choose if no other circuits with pullups were connected to the pin.

#### 4.2.5. HSO Pins, Control Outputs and Bus Pins

The control outputs and HSO pins have output buffers with the same output characteristics as those of the bus pins. Included in the category of control outputs are: TXD, RXD (in mode 0), PWM, CLKOUT, ALE, BHE, RD, and WR. The bus pins have 3 states: output high, output low, and high impedance input. As a high output, the pins are specified to source around 200  $\mu$ A to 2.4 volts, but the pins can source on the order of ten times that value in order to provide fast rise times. When used as a low output, the pins can sink around 2 mA at .45 volts, and considerably more as the voltage increases. When in the high impedance state, the pin acts as a capacitive load with a few microamps of leakage. Figure 4-9 shows the internal configuration of a bus pin.

### 4.3. ANALOG INTERFACE

Interfacing the 8096 to analog signal can be done in several ways. If the 8096 needs to measure an analog signal the A to D converter can be used. Creation of analog outputs can be done with either the PWM output or the HSO unit. There are differences between the A to D converter on the 8096 and the 8096BH. See the data sheets in Chapter 5.

#### 4.3.1. Analog Inputs

The 8096 can have 8 analog inputs and can convert one input at a time into a digital value. Each conversion takes 42 microseconds with a 12 MHz signal on XTAL1. The input signal is applied to one of the Port 0/Analog Channel inputs. Since there is no sample and hold on the A to D, the input signal must remain constant over the sampling period.

When a conversion takes place, the 8096 compares the external signal to that of its internal D to A. Based on the result of the comparison it adjusts the D to A and compares again. Each comparison takes 16 state times and requires the input to the comparator to be charged up. 10 comparisons are made during a conversion. An additional 8 states are used to load and store values. The total number of state times required is 168 for a 10-bit conversion. Attempting to do other than a 10-bit conversion is not recommended.

Since the capacitance of the comparator input is around 0.5pf, the sample and hold circuit must be able to charge a 5pf (10\*0.5pf) capacitor without a significant voltage change. To keep the effect of the sample and hold circuit below  $\pm 1/2$  lsb on a 10-bit converter, the voltage on the

sample and hold circuit may vary no more than 0.05% (1/2048).

The effective capacitance of the sample and hold must, therefore, be at least 1000pf or 0.01  $\mu$ f. If there is external leakage on the capacitor, its value must be increased to compensate for the leakage. At 10 $\mu$ A leakage, 2.5 mV (5/2048) will be lost from a 0.17  $\mu$ f capacitor in 42  $\mu$ s.

The capacitor connected externally to the pin should, therefore, be at least 0.2  $\mu$ f for best results. If the external signal changes slowly relative to 42  $\mu$ s, then a larger capacitor will work well and also filter out unwanted noise.

The converter is a 10-bit, successive approximation, ratio-metric converter, so the numerical value obtained from the conversion will be:

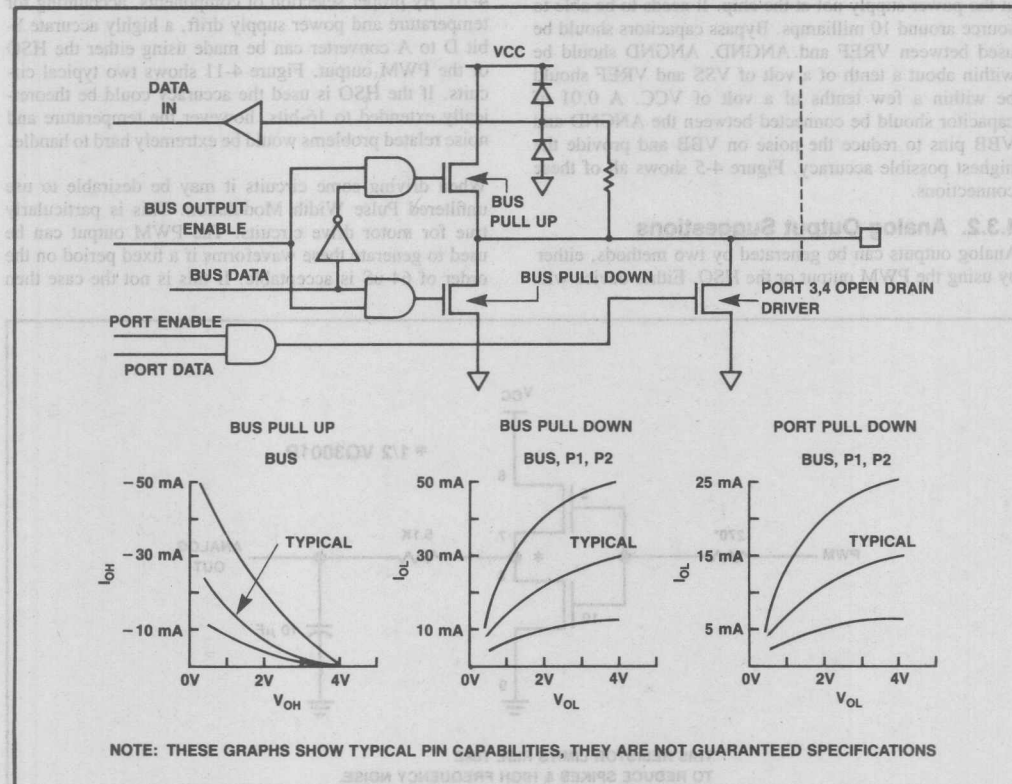


Figure 4-9. Bus and Port 3 and 4 Pins

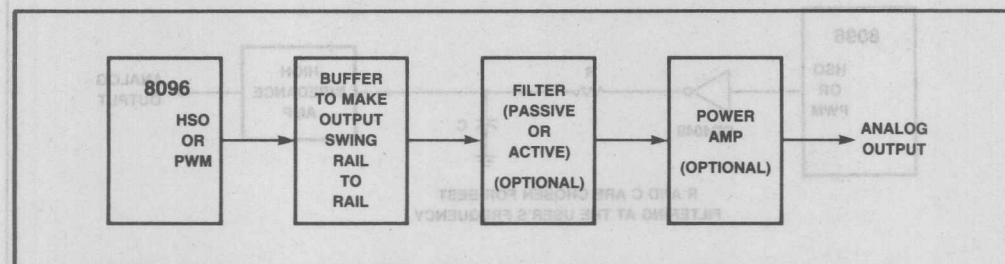


Figure 4-10. D/A Buffer Block Diagram

1023 \* (VIN-ANGND) / (VREF-ANGND)

It can be seen that the power supply levels strongly influence the absolute accuracy of the conversion. For this reason, it is recommended that the ANGND pin be tied to a clean ground, as close to the power supply as possible. VREF should be well regulated and used only for the A to D converter. If ratiometric information is desired, VREF can be connected to VCC, but this should be done at the power supply not at the chip. It needs to be able to source around 10 milliamps. Bypass capacitors should be used between VREF and ANGND. ANGND should be within about a tenth of a volt of VSS and VREF should be within a few tenths of a volt of VCC. A 0.01 uf capacitor should be connected between the ANGND and VBB pins to reduce the noise on VBB and provide the highest possible accuracy. Figure 4-5 shows all of these connections.

### 4.3.2. Analog Output Suggestions

Analog outputs can be generated by two methods, either by using the PWM output or the HSO. Either device will

generate a rectangular pulse train that varies in duty cycle and (for the HSO only) period. If a smooth analog signal is desired as an output, the rectangular waveform must be filtered.

In most cases this filtering is best done after the signal is buffered to make it swing from 0 to 5 volts since both of the outputs are guaranteed only to TTL levels. A block diagram of the type of circuit needed is shown in Figure 4-10. By proper selection of components, accounting for temperature and power supply drift, a highly accurate 8-bit D to A converter can be made using either the HSO or the PWM output. Figure 4-11 shows two typical circuits. If the HSO is used the accuracy could be theoretically extended to 16-bits, however the temperature and noise related problems would be extremely hard to handle.

When driving some circuits it may be desirable to use unfiltered Pulse Width Modulation. This is particularly true for motor drive circuits. The PWM output can be used to generate these waveforms if a fixed period on the order of 64 uS is acceptable. If this is not the case then

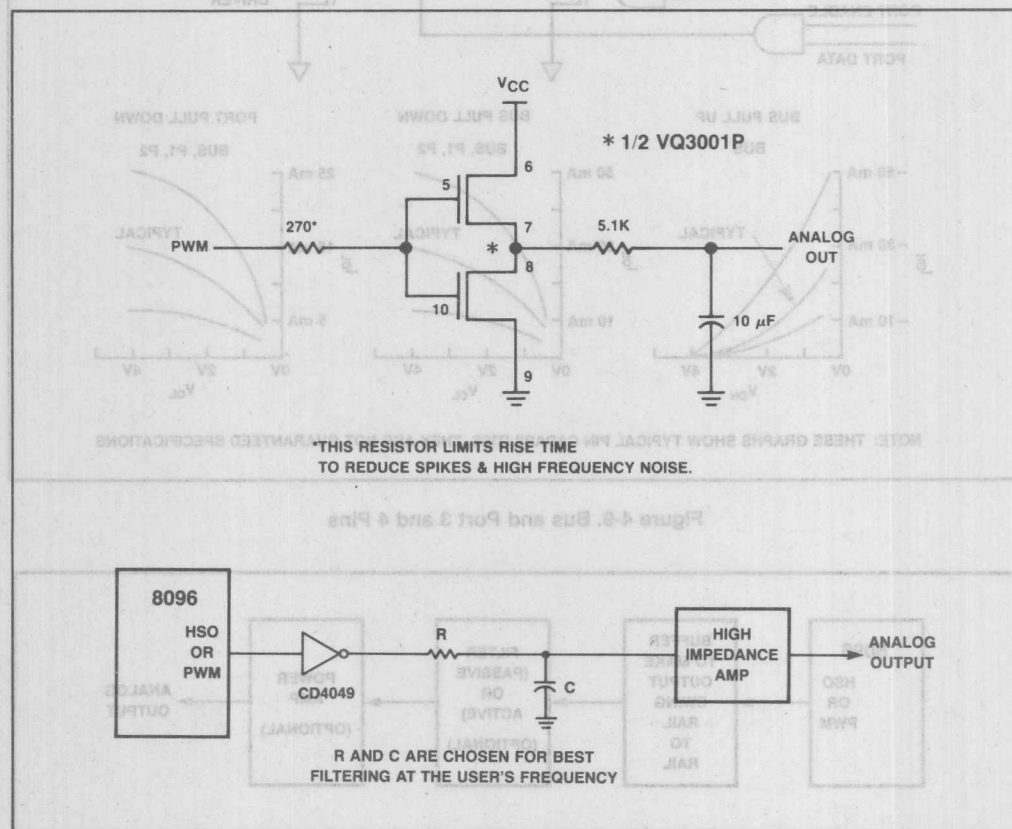


Figure 4-11. Buffer Circuits for D/A



the HSO unit can be used. The HSO can generate a variable waveform with a duty cycle variable in up to 65536 steps and a period of up to 131 milliseconds. Both of these outputs produce TTL levels.

#### 4.4. I/O TIMINGS

The I/O pins on the 8096 are sampled and changed at specific times within an instruction cycle. These times may differ between the 8096 and the 8096BH. The changes occur relative to the internal phases shown in figure 4-4. Note that the delay from XTAL1 to the internal clocks range from about 30 nS to 70 nS over process and temperature. Signals generated by internal phases are further delayed by 5 to 15 nS. The timings shown in this section are idealized; no propagation delay factors have been taken into account. Designing a system that depends on an I/O pin to change within a window of less than 50 nanoseconds using the information in this section is not recommended.

##### 4.4.1. HSO Outputs

Changes in the HSO lines are synchronized to Timer 1. All of the external HSO lines due to change at a certain value of a timer will change just prior to the incrementing of Timer 1. This corresponds to an internal change during Phase B every eight state times. From an external perspective the HSO pin should change just prior to the rising edge of CLKOUT and be stable by its falling edge. Information from the HSO can be latched on the CLKOUT falling edge. Internal events can occur anytime during the 8 state time window.

Timer 2 is synchronized to increment no faster than Timer 1, so there will always be at least one incrementing of Timer 1 while Timer 2 is at a specific value.

##### 4.4.2. HSI Input Sampling

The HSI pins are sampled internally once each state time. Any value on these pins must remain stable for at least 1 full state time to guarantee that it is recognized. The actual sample occurs at the end of Phase A, which, due to propagation delay, is just after the rising edge of CLKOUT. Therefore, if information is to be synchronized to the HSI it should be latched-in on CLKOUT falling. The time restriction applies even if the divide by eight mode is being used. If two events occur on the same pin within the same 8 state time window, only one of the events will be recorded. If the events occur on different pins they will always be recorded, regardless of the time difference. The 8 state time window, (ie. the amount of time during which Timer 1 remains constant), is stable to within about 20 nanoseconds. The window starts roughly around the rising edge of CLKOUT, however this timing is very approximate due to the amount of internal circuitry involved.

##### 4.4.3. Standard I/O Port Pins

Port 0 is different from the other digital ports in that it is actually part of the A to D converter. The port is sampled once every 8 state times, the same frequency at which the comparator is charged-up during an A to D conversion.

This 8 state times counter is not synchronized with Timer 1. If this port is used the input signal on the pin must be stable 8 state times prior to reading the SFR.

Port 1 and Port 2 have quasi-bidirectional I/O pins. When used as inputs the data on these pins must be stable one state time prior to reading the SFR. This timing is also valid for the input-only pins of Port 2 and is similar to the HSI in that the sample occurs just after the rising edge of CLKOUT. When used as outputs, the quasi-bidirectional pins will change state shortly after CLKOUT falls. If the change was from '0' to a '1' the low impedance pull-up will remain on for one state time after the change.

Ports 3 and 4 are addressed as off-chip memory-mapped I/O. The port pins will change state shortly after the rising edge of CLKOUT. When these pins are used as Ports 3 and 4 they are open drain, their structure is different when they are used as part of the bus. See Section 2.12.4.

#### 4.5. SERIAL PORT TIMINGS

The serial port on the 8096 was designed to be compatible with the 8051 serial port. Since the 8051 uses a divide by 2 clock and the 8096 uses a divide by 3, the serial port on the 8096 had to be provided with its own clock circuit to maximize its compatibility with the 8051 at high baud rates. This means that the serial port itself does not know about state times. There is circuitry which is synchronized to the serial port and to the rest of the 8096 so that information can be passed back and forth.

The baud rate generator is clocked by either XTAL1 or T2CLK, because T2CLK needs to be synchronized to the XTAL1 signal its speed must be limited to  $\frac{1}{16}$  that of XTAL1. The serial port will not function during the time between the consecutive writes to the baud rate register. Section 2.11.4 discusses programming the baud rate generator.

##### 4.5.1. Mode 0

Mode 0 is the shift register mode. The TXD pin sends out a clock train, while the RXD pin transmits or receives the data. Figure 4-12 shows the waveforms and timing. Note that the port starts functioning when a '1' is written to the REN (Receiver Enable) bit in the serial port control register. If REN is already high, clearing the RI flag will start a reception.

In this mode the serial port can be used to expand the I/O capability of the 8096 by simply adding shift registers. A schematic of a typical circuit is shown in Figure 4-13. This circuit inverts the data coming in, so it must be re-inverted in software. The enable and latch connections to the shift registers can be driven by decoders, rather than directly from the low speed I/O ports, if the software and hardware are properly designed.

##### 4.5.2. Mode 1 Timings

Mode 1 operation of the serial port makes use of 10-bit data packages, a start bit, 8 data bits and a stop bit. The

transmit and receive functions are controlled by separate shift clocks. The transmit shift clock starts when the baud rate generator is initialized, the receive shift clock is reset when a '1 to 0' transition (start bit) is received. The transmit clock may therefore not be in sync with the receive clock, although they will both be at the same frequency.

The TI (Transmit Interrupt) and RI (Receive Interrupt) flags are set to indicate when operations are complete. TI is set when the last data bit of the message has been sent, not when the stop bit is sent. If an attempt to send another byte is made before the stop bit is sent the port will hold off transmission until the stop bit is complete. RI is set

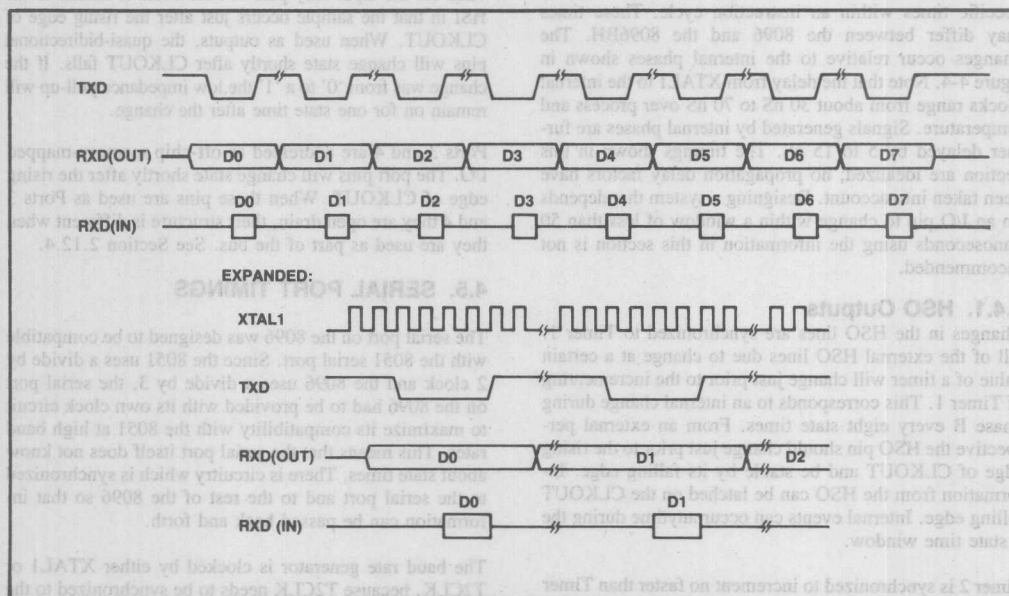


Figure 4-12. Serial Port Timings in Mode 0

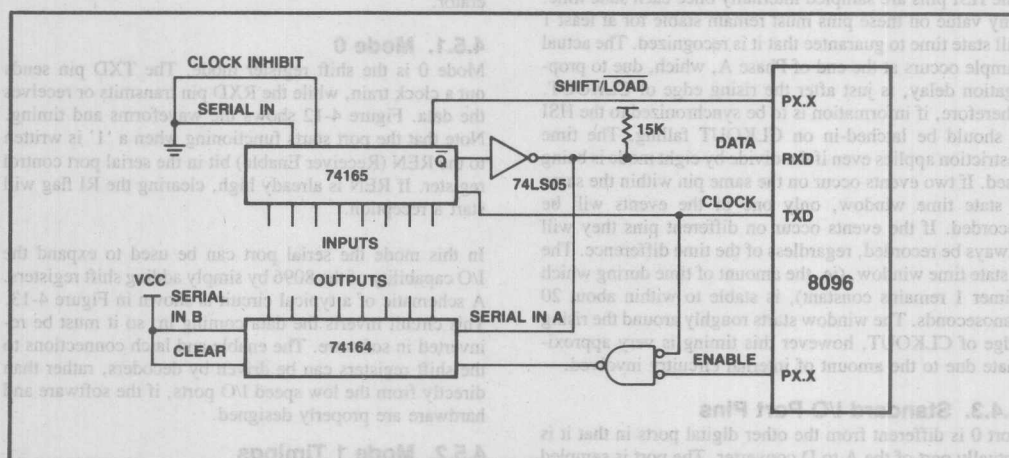


Figure 4-13. Mode 0 Serial Port Example



#### 4.5.3. Mode 2 and 3 Timings

Modes 2 and 3 operate in a manner similar to that of mode 1. The only difference is that the data is now made up of 9 bits, so 11-bit packages are transmitted and received. This means that TI and RI will be set on the 9th data bit rather than the 8th. The 9th bit can be used for parity or multiple processor communications (see section 2.11).

#### 4.6. BUS TIMING AND MEMORY INTERFACE

##### 4.6.1. Bus Functionality

The 8096 has a multiplexed (address/data) 16 bit bus. The 8096BH has a more flexible bus structure. See the 8096BH data sheet in Chapter 5. There are control lines provided

Tosc — Oscillator Period, one cycle time on XTAL1.

#### Timings The Memory System Must Meet

TLHYH — ALE low to READY high: Maximum time after ALE falls until READY is brought high to ensure no more wait states. If this time is exceeded unexpected wait states may result. Nominally  $1 \text{ TOSC} + 3 \text{ TOSC}^*$  number of wait states desired.

TLHYV — ALE low to READY VALID: Maximum time after ALE falls until READY must be valid. If this time is exceeded the part could malfunction necessitating a chip reset. Nominally 2 TOSC periods.

TYLYH — READY low to READY high: Maximum time the part can be in the not-ready state. If it is exceeded, the 8096 dynamic nodes which hold the current instruction may 'forget' how to finish the instruction.

TAVDV — ADDRESS valid to DATA valid: Maximum time that the memory has to output valid data after the 8096 outputs a valid address. Nominally, a maximum of 5 TOSC periods.

TRLDV —  $\overline{\text{READ}}$  low to DATA valid: Maximum time that the memory has to output data after READ goes low. Nominally, a maximum of 3 TOSC periods.

TRXDZ —  $\overline{\text{READ}}$  not low to DATA float: Time after  $\overline{\text{READ}}$  is no longer low until the memory must float the bus. The memory signal can be removed as soon as  $\overline{\text{READ}}$  is not low, and must be removed within the specified maximum time.

Nominally, a maximum of 1 TOSC period.

#### Timings the 8096 Will Provide

TCHCH — CLKOUT high to CLKOUT high: The period of CLKOUT and the duration of one state time. Always 3 TOSC average, but individual periods could vary by a few nanoseconds.

TCHCL — CLKOUT high to CLKOUT low: Nominally 1 TOSC period.

TCLLH — CLKOUT low to ALE high: A help in deriving other timings, typically plus or minus 5 to 10 ns.

TLLCH — ALE low to CLKOUT high: Used to derive other timings, nominally 1 TOSC period.

TLHLL — ALE high to ALE low: ALE pulse width. Useful in determining ALE rising edge to ADDRESS valid time. Nominally 1 TOSC period.

TAVLL — ADDRESS valid to ALE low: Length of time ADDRESS is valid before ALE falls. Important timing for address latch circuitry. Nominally 1 TOSC period.

TLLAX — ALE low to ADDRESS' invalid: Length of time ADDRESS is valid after ALE falls. Important timing for address latch circuitry. Nominally 1 TOSC period.

TLLRL — ALE low to  $\overline{\text{READ}}$  or  $\overline{\text{WRITE}}$  low: Length of time after ALE falls before  $\overline{\text{RD}}$  or  $\overline{\text{WR}}$  fall. Could be needed to ensure that proper memory decoding takes place before it is output enabled. Nominally 1 TOSC period.

TRLRH —  $\overline{\text{READ}}$  low to  $\overline{\text{READ}}$  high:  $\overline{\text{RD}}$  pulse width, nominally 1 TOSC period.

TRHLH —  $\overline{\text{READ}}$  high to ALE high: Time between  $\overline{\text{RD}}$  going inactive and next ALE, also used to calculate time between  $\overline{\text{RD}}$  inactive and next ADDRESS valid. Nominally 1 TOSC period.

TWLWH —  $\overline{\text{WRITE}}$  low to  $\overline{\text{WRITE}}$  high: Write pulse width, nominally 2 TOSC periods.

TQVWX — OUTPUT valid to  $\overline{\text{WRITE}}$  not low: time that the OUTPUT data is valid before  $\overline{\text{WR}}$  starts to go high. Nominally 2 TOSC periods.

TWXQX —  $\overline{\text{WRITE}}$  not low to OUTPUT not valid: Time that the OUTPUT data is valid after  $\overline{\text{WR}}$  starts to rise. Nominally 1 TOSC period.

TWXLH —  $\overline{\text{WRITE}}$  not low to ALE high: Time between write starting to rise and next ALE, also used to calculate the time between  $\overline{\text{WR}}$  starting to rise and next ADDRESS valid. Nominally 2 TOSC periods.

Figure 4-15. Timing Specification Explanations



to demultiplex the bus (ALE), indicate reads or writes ( $\overline{RD}$ ,  $\overline{WR}$ ), indicate if the access is for an instruction (INST), and separate the bus into high and low bytes ( $\overline{BHE}$ ,  $\overline{AD0}$ ). Section 2.3.5 contains an overview of the bus operation.

#### 4.6.2. Timing Specifications

Figure 4-14 shows the timing of the bus signals and data lines. Since this is a new part, the exact timing specifications are subject to change, please refer to the latest 8096 data sheet to ensure that your system is designed to the proper specifications. The major timing specifications are described in Figure 4-15.

#### 4.6.3. READY Line Usage

When the processor has to address a memory location that cannot respond within the standard specifications it is necessary to use the READY line to generate wait states. When the READY line is held low the processor waits in a loop for the line to come high. There is a maximum time that the READY line can be held low without risking a processor malfunction due to dynamic nodes that have not been refreshed during the wait states. This time is shown as TYLYH in the data sheet.

In most cases the READY line is brought low after the address is decoded and it is determined that a wait state is needed. It is very likely that some addresses, such as those addressing memory mapped peripherals, would need wait states, and others would not. The READY line must be stable within the TLLYV specification after ALE falls (or the TYVCL before CLKOUT falls) or the processor could lock-up. There is no requirement as to when READY may go high, as long as the maximum READY low time (TYLYH) is not violated. To ensure that only one wait state is inserted it is necessary to bring READY high TLLYH after the falling edge of ALE.

#### 4.6.4. INST Line Usage

The INST (Instruction) line is high during the output of

an address that is for an instruction stream fetch. It is low during the same time for any other memory access. At any other time it is not valid. This pin is not present on the 48-pin versions. The INST signal can be used with a logic analyzer to debug a system. In this way it is possible to determine if the fetch was for instructions or data, making the task of tracing the program much easier.

#### 4.6.5. Address Decoding

The multiplexed bus of the 8096 must be demultiplexed before it can be used. This can be done with 2 74LS373 transparent latches. As explained in section 2.3.5, the latched address signal will be referred to as MA0 through MA15. (Memory Address), and the data lines will be called MD0 through MD15, (Memory Data).

Since the 8096 can make accesses to memory for either bytes or words it is necessary to have a way of determining the type of access desired. The  $\overline{BHE}$  and MA0 lines are used for this purpose.  $\overline{BHE}$  must be latched, as it is valid only when the address is valid. The memory system is typically set up as 32K by 16, instead of 64K by 8. When the  $\overline{BHE}$  line is low, the upper byte is enabled. When MA0 is low, the lower byte is enabled. When MA0 is low and  $\overline{BHE}$  is low, both bytes are enabled.

When external RAM and EPROM are both used in the system the control logic can be simplified a little to some of the addresses. The 8096 will always output  $\overline{BHE}$  to indicate if a read is of the high byte or the low byte, but it discards the byte it is not going to use. It is therefore possible to use the  $\overline{BHE}$  and MA0 lines only to control memory writes, and to ignore these lines during memory reads. Figure 4-16 and 4-17 show block diagrams of two memory systems, an external EPROM only system and a RAM/ROM system.

#### 4.6.6. System Verification Example

To verify that a system such as the one in Figure 4-17 will work with the 8096, it is necessary to check all of the

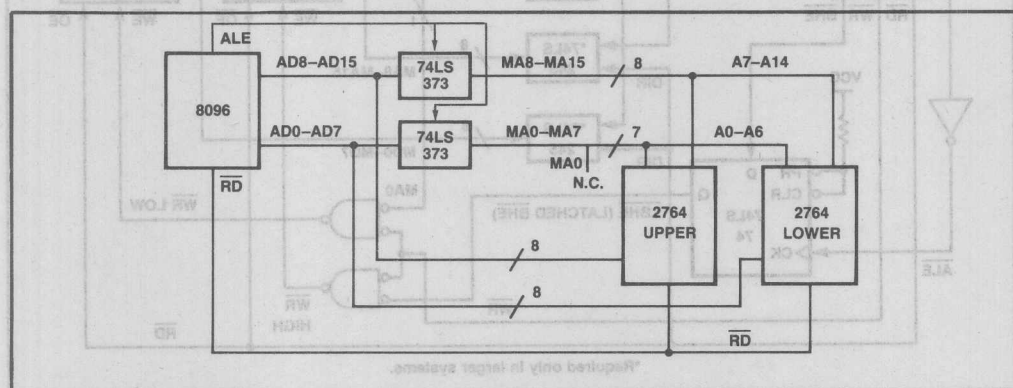


Figure 4-16. Memory Wiring Example—EPROM Only System

The timings or signals that the processor and memory use are effected by the latch and buffer circuitry. The timings of the signal provided by the processor are delayed by various amounts of time. Similarly, the signals coming back from the memory are also delayed. The calculations involved in verifying this system follow:

The address lines are delayed by passing them through the 74LS373s, this delay is specified at 18ns after Address is valid or 30ns after ALE is high. Since the signal may be limited by either the ALE timing or the Address timing, these two cases must be considered.

Minimum ALE pulse width =  $\frac{T_{osc} - 10}{f_{osc}}$  (TLHLL)

Minimum Addr set-up to ALE falling = T<sub>osc</sub>-20 (TAVLL)

Total delay from 8096 Address stable to MA (Memory Address) stable would be:

|                         |       |             |
|-------------------------|-------|-------------|
| ALE delay from address  | -10   |             |
| 74LS373 clock to output | 30    |             |
|                         | <hr/> |             |
|                         | 20    | nanoseconds |

74LS373 Data Valid to Data Output = 18 nanoseconds

In the worst case, the delay in Address valid is controlled by ALE and has a value of 20 nanoseconds.

The  $\overline{\text{RD}}$  low to Data valid specification (TRLDV) is 3 Tasc-50, (200 ns at 12 MHz). The 74LS245 is enabled by  $\overline{\text{RD}}$  and has a delay of 40 ns from enable. The enable delay is clearly not a problem.

The 74LS245 is enabled for write, except during a read, so there is no enable delay to consider for write operations.



The Data In to Data Out delay of the 74LS245 is 12 ns.

#### Delay of $\overline{WR}$ signal to memory — 15 nanoseconds

Latched  $\overline{BHE}$  is delayed by the inverter on ALE and the 74LS74.

74LS04 delay (Output low to high) = 22

74LS74 delay (Clock to Output) = 40

Delay of Latched  $\overline{BHE}$  from ALE falling = 62  
nanoseconds

The 74LS74 requires data valid for 20 ns prior to the clock, the 8096 will have  $\overline{BHE}$  stable Tsc-20 ns (TAVLL, 63 ns at 12 MHz) prior to ALE falling. There is no problem here.

MA0 is valid prior to ALE falling, since the 20 ns Address Delay is less than TAVLL.

$\overline{WR}$  will fall no sooner than Tsc-20 ns (TLLRL, 63 ns at 12 MHz) after ALE goes low. It will therefore be valid just after the Latched  $\overline{BHE}$  is valid, so it is the controlling signal.

$\overline{WR}$  High and  $\overline{WR}$  Low are valid 15 ns after MA0, Latched  $\overline{BHE}$  and  $\overline{WR}$  are valid. Since  $\overline{WR}$  is the last signal to go valid, the delay of  $\overline{WR}$  (High and Low) to memory is 15 ns.

**Delay Summary** — Address Delay = 20 ns  
Data Delay = 12 ns  
 $\overline{WR}$  Delay = 15 ns  
RD Delay = 0 ns

#### Characteristics of a 12 MHz 8096 system with latches:

Required by system:

Address valid to Data in;

TAVDV : 386.6 ns max. (5 Tsc-30)  
Address Delay : — 20.0 ns maximum  
Data Delay : — 12.0 ns maximum  
354.6 ns maximum

Read low to Data in;

TRLDV : 200.0 ns max. (3 Tsc-50)  
RD Delay : — 00.0 ns maximum  
Data Delay : — 12.0 ns maximum  
188.0 ns maximum

Provided by System:

Address valid to Control;

TLLRL : 63.3 ns min. (Tsc-20)  
TAVLL : 63.3 ns min. (Tsc-20)  
Address Delay : — 20.0 ns maximum  
 $\overline{WR}$  Delay : — 00.0 ns minimum (no spec)  
101.6 ns minimum

Write Pulse Width;

TWLWH : 151.6 ns min. (2 Tsc-15)  
Rising  $\overline{WR}$  Delay : — 15.0 ns maximum  
Falling  $\overline{WR}$  Delay : — 00.0 ns minimum (no spec)  
146.6 ns minimum

Data Setup to  $\overline{WR}$  rising;

TQVWX : 136.6 ns min. (2 Tsc-30)  
Data Delay : — 12.0 ns maximum  
 $\overline{WR}$  Delay : — 00.0 ns minimum (no spec)  
124.6 ns minimum

Data Hold after  $\overline{WR}$ ;

TWXQX : 58.3 ns min. (Tsc-25)  
Data Delay : — 0.0 ns minimum (no spec)  
 $\overline{WR}$  Delay : — 15.0 ns maximum  
43.3 ns minimum

The two memory devices which are expected to be used most often with the 8096 are the 2764 EPROM and the 2128 RAM. The system verification for the 2764 is simple.

#### 2764 Tac

(Address valid to Output) < Address valid to Data in  
250 ns < 354 ns O.K.

#### 2764 Toe

(Output Enable to Output) < Read low to Data in  
100 ns < 188 ns O.K.

These calculations assume no address decoder delays and no delays on the RD (OE) line. If there are delays in these signals the delays must be added to the 2764's timing.

The read calculations for the 2128 are similar to those for the 2764.

2128-20 Tac < Address valid to Data in  
200 ns < 354 ns O.K.

2128-20 Toe < Read low to Data in  
65 ns < 188 ns O.K.

The write calculations are a little more involved, but still straight-forward.

2128 Twp (Write Pulse) < Write Pulse Width  
100 ns < 146 ns O.K.

2128 Tds (Data Setup) < Data Setup to  $\overline{WR}$  rising  
65 ns < 124 ns O.K.

2128 Tdh (Data Hold) < Data Hold after  $\overline{WR}$   
0 ns < 43 ns

All of the above calculations have been done assuming that no components are in the circuit except for those shown in Figure 4-17. If additional components are added, as may be needed for address decoding or memory bank switching, the calculations must be updated to reflect the actual circuit.

#### 4.6.7. I/O Port Reconstruction

When a single-chip system is being designed using a multiple chip system as a prototype, it may be necessary to reconstruct I/O ports 3 and 4 using a memory-mapped I/O technique. The circuit shown in Figure 4-18 provides this function on the iSBE-96 emulator board. It can be attached to any 8096 system which has the required address decoding and bus demultiplexing.

The output circuitry is basically just a latch that operates when IFFEH or IFFFFH are placed on the MA lines. The

inverters surrounding the latch create an open-collector output to emulate the open-drain output found on the 8096. The 'reset' line is used to set the ports to all 1's when the 8096 is reset. It should be noted that the voltage and current characteristics of this port will differ from those of the 8096, but the basic functionality will be the same.

The input circuitry is just a bus transceiver that is addressed at IFFEH or IFFFFH. If the ports are going to be used for either input or output, but not both, some of the circuitry can be eliminated.

#### 4.7. NOISE PROTECTION TIPS

Designing controllers differs from designing other computer equipment in the area of noise protection. A microcontroller circuit under the hood of a car, in a photocopier, CRT terminal, or a high speed printer is subject to many types of electrical noise. Noise can get to the processor directly through the power supply, or it can be induced onto the board by electromagnetic fields. It is also possible for the pc board to find itself in the path of electrostatic discharges. Glitches and noise on the pc board can cause the processor to act unpredictably, usually by

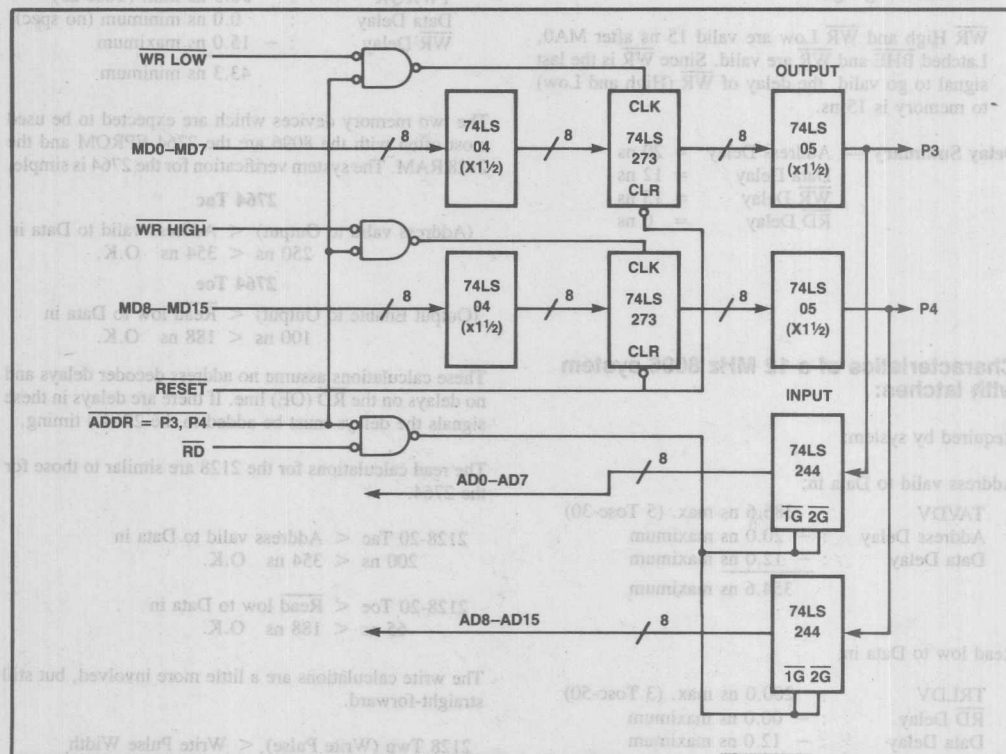


Figure 4-18. I/O Port Reconstruction



changing either the memory locations or the program counter.

There are both hardware and software solutions to noise problems, but the best solution is good design practice and a few ounces of prevention. The 8096 has a watchdog timer which will reset the part if it fails to execute the software properly. The software should be set up to take advantage of this feature.

It is also recommended that unused areas of code be filled with NOPs and periodic jumps to an error routine or RST (reset chip) instructions. This is particularly important in the code around lookup tables, since if lookup tables are executed all sorts of bad things can happen. Wherever space allows, each table should be surrounded by 7 NOPs (the longest 8096 instruction has 7 bytes) and a RST or jump to error routine instruction. This will help to ensure a speedy recovery should the processor have a glitch in the program flow.

Many hardware solutions exist for keeping pc board noise to a minimum. Ground planes, gridded ground and VCC structures, bypass capacitors, transient absorbers and power busses with built-in capacitors can all be of great help. It is much easier to design a board with these features

than to try to retrofit them later. Proper pc board layout is probably the single most important and, unfortunately, least understood aspect of project design. Minimizing loop areas and inductance, as well as providing clean grounds are very important. More information on protecting against noise can be found in the Intel Application Note AP-125, "Designing Microcontroller Systems For Noisy Environments."

#### 4.8. PACKAGING PINOUTS AND ENVIRONMENT

The MCS-96 family of products is offered in many versions. They are available in 48-pin or 68-pin packages, Romless, with EPROM or with ROM, and with or without an A to D converter. A summary of the available options is shown in Figure 4-19.

The 48-pin versions are available in a ceramic 48-pin DIP (Dual In-Line) package.

The 68-pin versions are available in a ceramic pin grid array, and a plastic leaded chip carrier.

Specifications for the various members of the MCS-96 family are contained in the next chapter.

|                | ROMLESS |        | WITH ROM |        | WITH EPROM |        |
|----------------|---------|--------|----------|--------|------------|--------|
|                | 68-pin  | 48-pin | 68-pin   | 48-pin | 68-pin     | 48-pin |
| Without A to D | 8096    | 8094   | 8396     | 8394   | 8796       | 8794   |
| With A to D    | 8097    | 8095   | 8397     | 8395   | 8797       | 8795   |

Figure 4-19. The MCS®-96 Family of Products









■ **839X:** an 809X with 8

- High Speed Pulse I/O
- 10-bit A/D Converter
- 6.5  $\mu$ sec 16 x 16 Multiply
- 6.5  $\mu$ sec 32/16 Divide
- 8 Interrupt Sources
- Pulse-Width Modulated Output
- 232 Byte Register File
- Memory-to-Memory Architecture
- Full Duplex Serial Port
- Five 8-bit I/O Ports
- Watchdog Timer
- Four 16-bit Software Timers

control functions.

The CPU supports bit, byte, and word operations. 32-bit double-words are supported for a subset of the instruction set. With a 12 MHz input frequency the 8096 can do a 16-bit addition in 1.0  $\mu$ sec and a 16 x 16-bit multiply or 32/16-bit divide in 6.5  $\mu$ sec. Instruction execution times average 1 to 2  $\mu$ sec in typical applications.

Four high-speed trigger inputs are provided to record the times at which external events occur. Six high-speed pulse generator outputs are provided to trigger external events at preset times. The high-speed output unit can simultaneously perform timer functions. Up to four such 16-bit Software Timers can be in operation at once.

on-chip A/D Converter converts up to 4 (in the 48-pin version) or 8 (in the 68-pin version) analog input channels to 10-bit digital values. This feature is only available on the 8095-90/8395-90 and 8097-90/8397-90.

Also provided on-chip are a serial port, a watchdog timer, and a pulse-width modulated output signal.

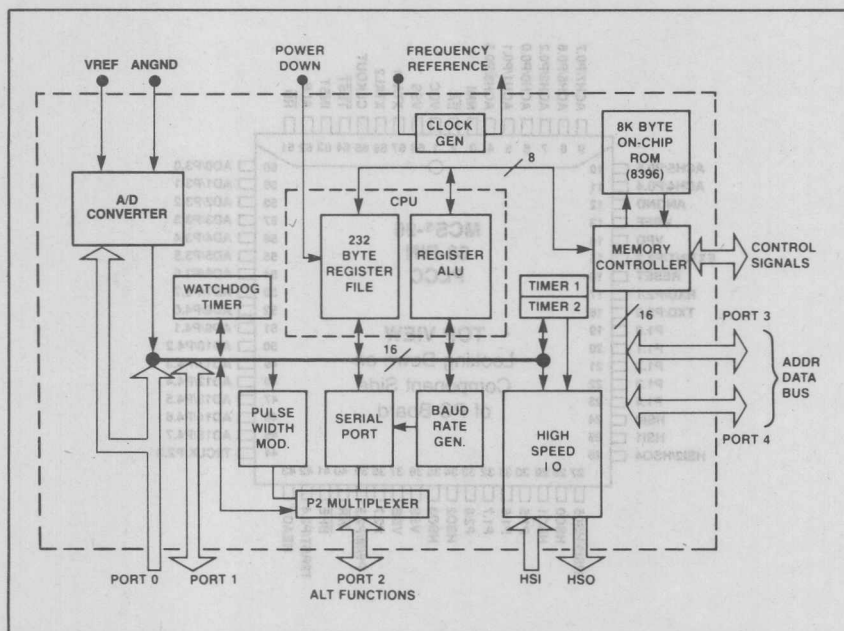


Figure 3. 68-pin PLC Package

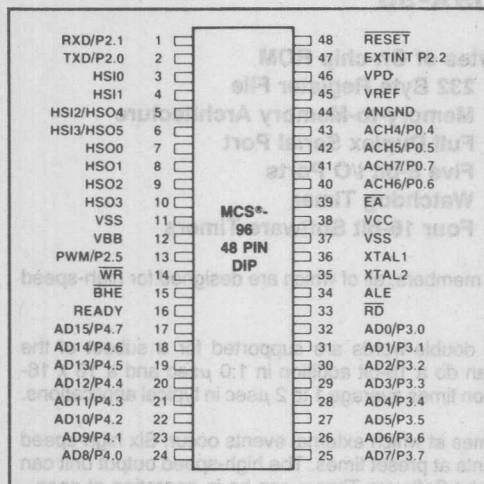


Figure 2. 48-Pin Package

Figure 1 shows a block diagram of the MCS-96 parts generally referred to as the 8096. The 8096 is available in 48-pin and 68-pin packages, with and without A/D, and with and without on-chip ROM. The MCS 96 numbering system is shown below:

| OPTIONS                |         | 68-PIN  | 48-PIN  |
|------------------------|---------|---------|---------|
| DIGITAL I/O            | ROMLESS | 8096-90 | 8094-90 |
|                        | ROM     | 8396-90 | 8394-90 |
| ANALOG AND DIGITAL I/O | ROMLESS | 8097-90 | 8095-90 |
|                        | ROM     | 8397-90 | 8395-90 |

Figures 2, 3 and 4 show the pinouts for the 48- and 68-pin packages. The 48-pin version is offered in Dual-In-Line package while the 68-pin version comes in a Plastic Leaded Chip Carrier and a Pin Grid Array.

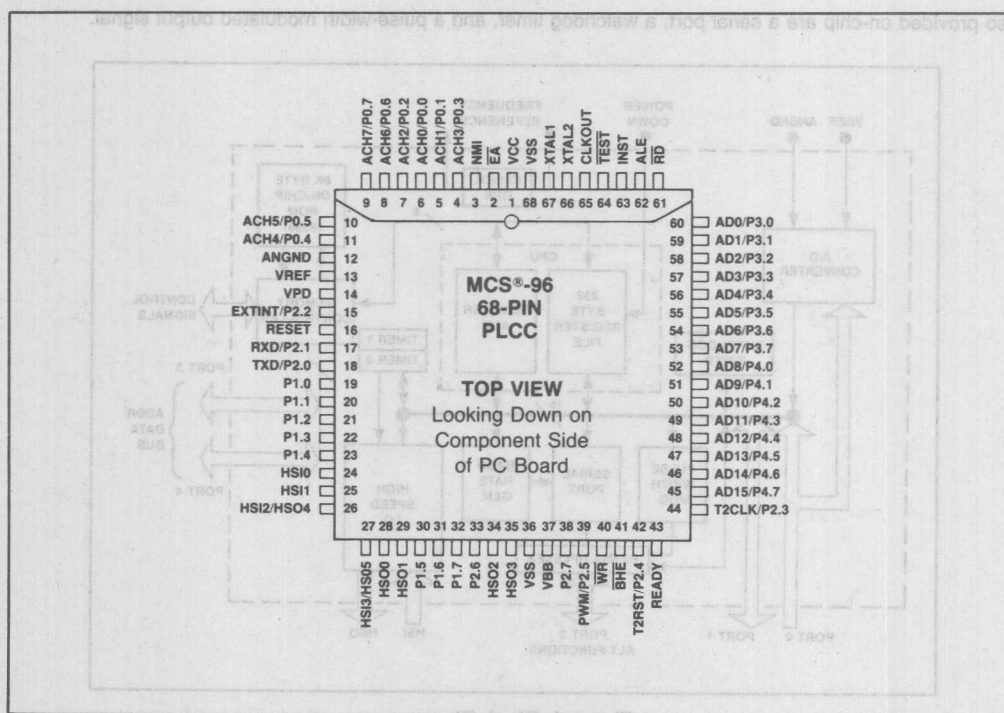


Figure 3. 68-Pin PLCC Package

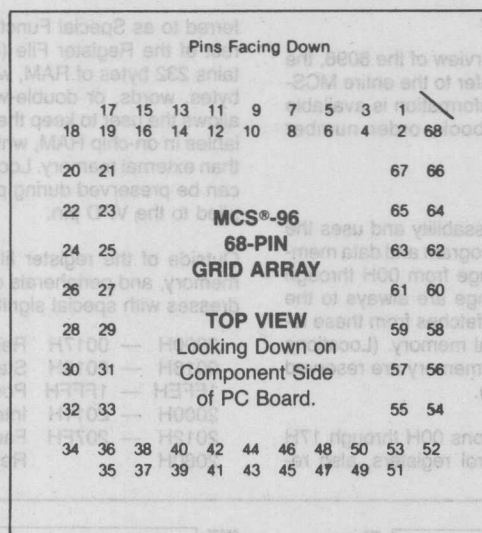


Figure 4. Pin Grid Array

| PGA | PLCC | Description | PGA | PLCC | Description | PGA | PLCC | Description |
|-----|------|-------------|-----|------|-------------|-----|------|-------------|
| 1   | 9    | ACH7/P0.7   | 24  | 54   | AD6/P3.6    | 47  | 31   | P1.6        |
| 2   | 8    | ACH6/P0.6   | 25  | 53   | AD7/P3.7    | 48  | 30   | P1.5        |
| 3   | 7    | ACH2/P0.2   | 26  | 52   | AD8/P4.0    | 49  | 29   | HSO.1       |
| 4   | 6    | ACH0/P0.0   | 27  | 51   | AD9/P4.1    | 50  | 28   | HSO.0       |
| 5   | 5    | ACH1/P0.1   | 28  | 50   | AD10/P4.2   | 51  | 27   | HSO.5/HSI.3 |
| 6   | 4    | ACH3/P0.3   | 29  | 49   | AD11/P4.3   | 52  | 26   | HSO.4/HSI.2 |
| 7   | 3    | NMI         | 30  | 48   | AD12/P4.4   | 53  | 25   | HSI.1       |
| 8   | 2    | EA          | 31  | 47   | AD13/P4.5   | 54  | 24   | HSI.0       |
| 9   | 1    | VCC         | 32  | 46   | AD14/P4.6   | 55  | 23   | P1.4        |
| 10  | 68   | VSS         | 33  | 45   | AD15/P4.7   | 56  | 22   | P1.3        |
| 11  | 67   | XTAL1       | 34  | 44   | T2CLK/P2.3  | 57  | 21   | P1.2        |
| 12  | 66   | XTAL2       | 35  | 43   | READY       | 58  | 20   | P1.1        |
| 13  | 65   | CLKOUT      | 36  | 42   | T2RST/P2.4  | 59  | 19   | P1.0        |
| 14  | 64   | TEST        | 37  | 41   | BHE         | 60  | 18   | TXD/P2.0    |
| 15  | 63   | INST        | 38  | 40   | WR          | 61  | 17   | RXD/P2.1    |
| 16  | 62   | ALE         | 39  | 39   | PWM/P2.5    | 62  | 16   | RESET       |
| 17  | 61   | RD          | 40  | 38   | P2.7        | 63  | 15   | EXTINT/P2.2 |
| 18  | 60   | AD0/P3.0    | 41  | 37   | VBB         | 64  | 14   | VPD         |
| 19  | 59   | AD1/P3.1    | 42  | 36   | VSS         | 65  | 13   | VREF        |
| 20  | 58   | AD2/P3.2    | 43  | 35   | HSO.3       | 66  | 12   | ANGND       |
| 21  | 57   | AD3/P3.3    | 44  | 34   | HSO.2       | 67  | 11   | ACH4/P0.4   |
| 22  | 56   | AD4/P3.4    | 45  | 33   | P2.6        | 68  | 10   | ACH5/P0.5   |
| 23  | 55   | AD5/P3.5    | 46  | 32   | P1.7        |     |      |             |

## FUNCTIONAL OVERVIEW

The following section is an overview of the 8096, the generic part number used to refer to the entire MCS-96 product family. Additional information is available in the Microcontroller Handbook, order number 210918-003.

## CPU Architecture

The 8096 has 64 Kbyte addressability and uses the same address space for both program and data memory, except in the address range from 00H through 0FFH. Data fetches in this range are always to the Register File, while instruction fetches from these locations are directed to external memory. (Locations 00H through 0FFH in external memory are reserved for Intel development systems).

Within the Register File, locations 00H through 17H are register mapped I/O control registers, also re-

ferred to as Special Function Registers (SFRs). The rest of the Register File (018H through 0FFH) contains 232 bytes of RAM, which can be referenced as bytes, words, or double-words. This register space allows the user to keep the most frequently-used variables in on-chip RAM, which can be accessed faster than external memory. Locations 0F0H through 0FFH can be preserved during power down if power is applied to the VPD pin.

Outside of the register file, program memory, data memory, and peripherals can be intermixed. The addresses with special significance are:

|                |                            |
|----------------|----------------------------|
| 0000H — 0017H  | Register-mapped I/O (SFRs) |
| 0018H — 0019H  | Stack Pointer              |
| 1FFEh — 1FFFFH | Ports 3 and 4              |
| 2000H — 2011H  | Interrupt Vectors          |
| 2012H — 207FH  | Factory Test Code          |
| 2080H          | Reset Location             |

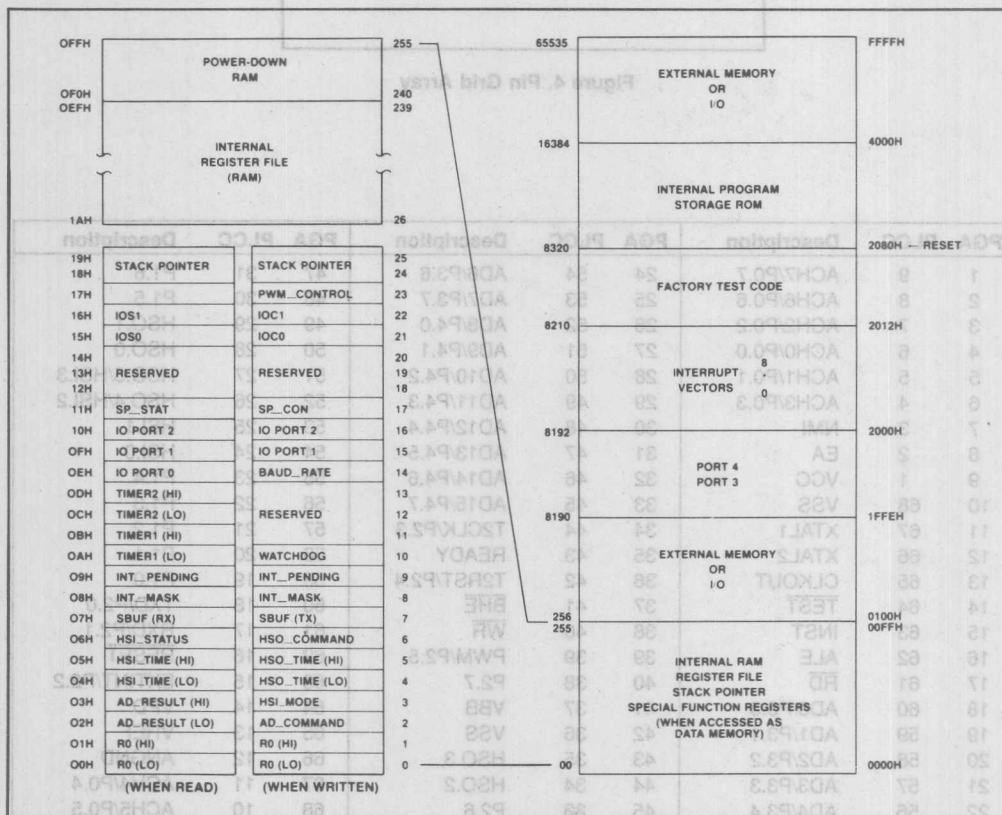


Figure 5. Memory Map



The 839x carries 8K bytes of on-chip ROM, occupying addresses 2000H through 3FFFH. Instruction or data fetches from these addresses access the on-chip ROM if the  $\overline{EA}$  pin is externally held at a logical 1. If the  $\overline{EA}$  pin is at a logical 0 these addresses access off-chip memory.

A memory map for the MCS-96 product family is shown in Figure 5.

The RALU (Register/ALU) section consists of a 17-bit ALU, the Program Status Word, the Program Counter, and several temporary registers. A key feature of the 8096 is that it does not use an accumulator. Rather, it operates directly on any register in the Register File. Being able to operate directly on data in the Register File without having to move it into and out of an accumulator results in a significant improvement in execution speed.

In addition to the normal arithmetic and logical functions, the MCS-96 instruction set provides the following special features:

- 6.5  $\mu$ s Multiply and Divide
- Multiple Shift Instructions
- 3 Operand Instructions
- Normalize Instruction
- Software Reset Instruction

All operations on the 8096 take place in a set number of "State Times." The 8096 uses a three-phase in-

ternal clock, so each state time is 3 oscillator periods. With a 12 MHz clock, each state time requires 0.25 microseconds.

### High Speed I/O Unit (HSIO)

The HSIO unit consists of the High Speed Input Unit (HSI), the High Speed Output Unit (HSO), one counter and one timer. "High Speed" denotes that the units can perform functions related to the timers without CPU intervention. The HSI records times when events occur and the HSO triggers events at preprogrammed times.

All actions within the HSIO unit are synchronized to the timers. The two 16-bit timer/counter registers in the HSIO unit are cleared on chip reset and can be programmed to generate an interrupt on overflow. The Timer 1 register is automatically incremented every 8 state times (every 2.0 microseconds, with a 12 MHz clock). The Timer 2 register can be programmed to count transitions on either the T2CLK pin or HSI.1 pin. It is incremented on both positive and negative edges of the selected input line. In addition to being cleared by reset, Timer 2 can also be cleared in software or by signals from input pins T2RST or HSI.0. Neither of these timers is required for the Watchdog timer or the serial port.

The High Speed Input (HSI) unit can detect transitions on any of its 4 input lines. When one occurs it records the time (from Timer 1) and which input lines made

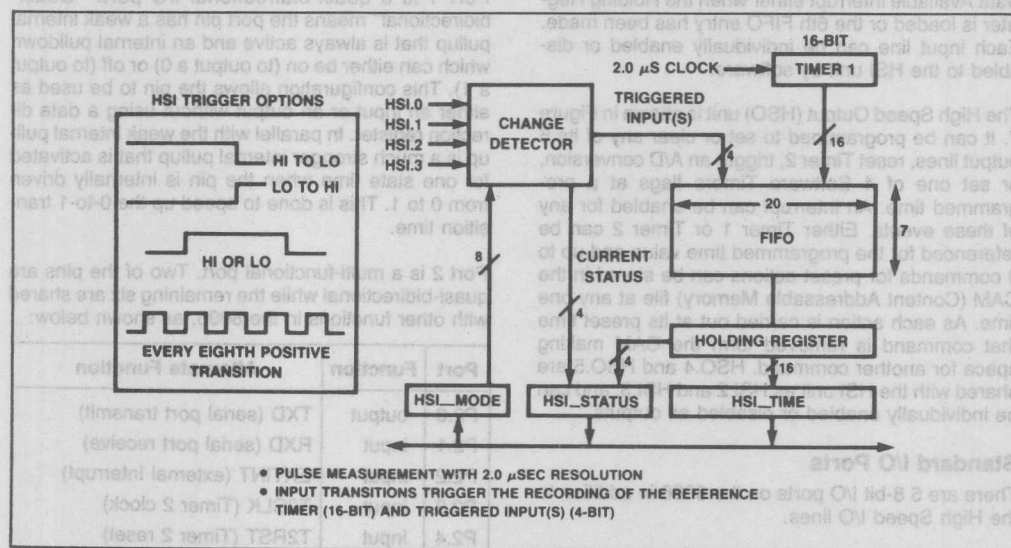
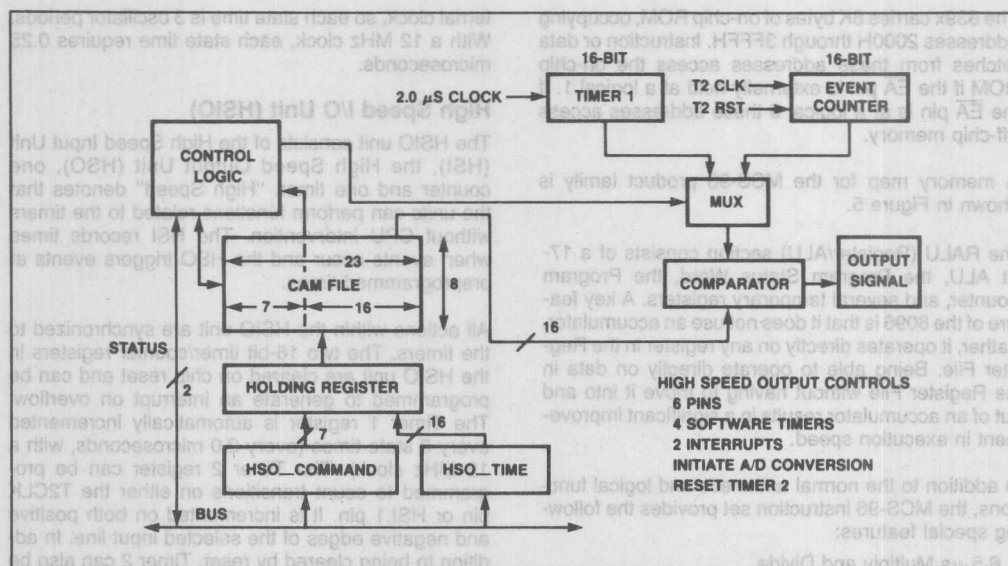


Figure 6. High Speed Input Unit



**Figure 7. High Speed Output Unit**

the transition. This information is recorded with 2 microsecond resolution and stored in an 8-level FIFO. The unit can be programmed to look for four types of events, as shown in Figure 6. It can activate the HSI Data Available interrupt either when the Holding Register is loaded or the 6th FIFO entry has been made. Each input line can be individually enabled or disabled to the HSI unit by software.

The High Speed Output (HSO) unit is shown in Figure 7. It can be programmed to set or clear any of its 6 output lines, reset Timer 2, trigger an A/D conversion, or set one of 4 Software Timers flags at a programmed time. An interrupt can be enabled for any of these events. Either Timer 1 or Timer 2 can be referenced for the programmed time value and up to 8 commands for preset actions can be stored in the CAM (Content Addressable Memory) file at any one time. As each action is carried out at its preset time that command is removed from the CAM making space for another command. HSO.4 and HSO.5 are shared with the HSI unit as HSI.2 and HSI.3, and can be individually enabled or disabled as outputs.

### Standard I/O Ports

There are 5 8-bit I/O ports on the 8096 in addition to the High Speed I/O lines.

Port 0 is an input-only port which shares its pins with the analog inputs to the A/D Converter. The port can

be read digitally and/or, by writing to the A/D Command Register, one of the lines can be selected as the input to the A/D Converter.

Port 1 is a quasi-bidirectional I/O port. "Quasi-bidirectional" means the port pin has a weak internal pullup that is always active and an internal pulldown which can either be on (to output a 0) or off (to output a 1). This configuration allows the pin to be used as either an input or an output without using a data direction register. In parallel with the weak internal pullup is a much stronger internal pulldown that is activated for one state time when the pin is internally driven from 0 to 1. This is done to speed up the 0-to-1 transition time.

Port 2 is a multi-functional port. Two of the pins are quasi-bidirectional while the remaining six are shared with other functions in the 8096, as shown below:

| Port | Function | Alternate Function           |
|------|----------|------------------------------|
| P2.0 | output   | TXD (serial port transmit)   |
| P2.1 | input    | RXD (serial port receive)    |
| P2.2 | input    | EXTINT (external Interrupt)  |
| P2.3 | input    | T2CLK (Timer 2 clock)        |
| P2.4 | input    | T2RST (Timer 2 reset)        |
| P2.5 | output   | PWM (pulse-width modulation) |

Ports 3 and 4 are bi-directional I/O ports with open drain outputs. These pins are also used as the multiplexed address/data bus when accessing external memory, in which case they have strong internal pull-ups. The internal pullups are only used during external memory read or write cycles when the pins are outputting address or data bits. At any other time, the internal pullups are disabled.

### Serial Port

The serial port is compatible with the MCS-51 family (8051, 8031 etc.) serial port. It is full duplex, and receive-buffered. There are 3 asynchronous modes and 1 synchronous mode of operation for the serial port. The asynchronous modes allow for 8 or 9 bits of data with even parity optionally inserted for one of the data bits. Selective interrupts based on the 9th data bit are available to support interprocessor communication.

Baud rates in all modes are determined by an independent 16-bit on-chip baud rate generator. Either the XTAL1 pin or the T2CLK pin can be used as the input to the baud rate generator. The maximum baud rate in the asynchronous mode is 187.5 KBaud.

### Pulse Width Modulator (PWM)

The PWM output shares a pin with port bit P2.5. When the PWM output is selected, this pin outputs a pulse train having a fixed period of 256 state times, and a programmable width of 0 to 255 state times. The width is programmed by loading the desired value, in state times, to the PWM Control Register.

### A/D Converter

The analog-to-digital converter is a 10-bit, successive approximation converter. It has a fixed conversion time of 168 state times, (42 microseconds with a 12 MHz clock). The analog input must be in the range of 0 to VREF (normally, VREF = 5V). This input can be selected from 8 analog input lines, which connect to the same pins as Port 0. A conversion can be initiated either by setting a control bit in the A/D Command register, or by programming the HSO unit to trigger the conversion at some specified time.

### Interrupts

The 8096 has 20 interrupt sources which vector through 8 locations. A 0-to-1 transition from any of the sources sets a corresponding bit in the Interrupt Pending register. The content of the Interrupt Mask register determines if a pending interrupt will be

served or not. If it is to be serviced, the CPU pushes the current Program Counter onto the Stack and reloads it with the vector corresponding to the desired interrupt. The interrupt vectors are located in addresses 2000H through 2011H, as shown in Figure 8.

| Source         | Vector Location |            | Priority       |
|----------------|-----------------|------------|----------------|
|                | (High Byte)     | (Low Byte) |                |
| Software       | 2011H           | 2010H      | Not Applicable |
| Extint         | 200FH           | 200EH      | 7 (Highest)    |
| Serial Port    | 200DH           | 200CH      | 6              |
| Software       | 200BH           | 200AH      | 5              |
| Timers         |                 |            |                |
| HSI.0          | 2009H           | 2008H      | 4              |
| High Speed     | 2007H           | 2006H      | 3              |
| Outputs        |                 |            |                |
| HSI Data       | 2005H           | 2004H      | 2              |
| Available      |                 |            |                |
| A/D Conversion | 2003H           | 2002H      | 1              |
| Complete       |                 |            |                |
| Timer Overflow | 2001H           | 2000H      | 0 (Lowest)     |

Figure 8. Interrupt Vectors

At the end of the interrupt routine the RET instruction pops the program counter from the stack and execution continues where it left off. It is not necessary to store and replace registers during interrupt routines as each routine can be set up to use a different section of the register file. This feature of the architecture provides for very fast context switching.

While the 8096 has a single priority level in the sense that any interrupt may be itself be interrupted, a priority structure exists for resolving simultaneously pending interrupts, as indicated in Figure 8. Since the interrupt pending and interrupt mask registers can be manipulated in software, it is possible to dynamically alter the interrupt priorities to suit the users' software.

### Watchdog Timer

The watchdog timer is a 16-bit counter which, once started, is incremented every state time. After 16 milliseconds, if not cleared, it will overflow, pulling down the RESET pin for two state times, causing the system to be reinitialized. This feature is provided as a means of graceful recovery from a software upset. The counter must be cleared by the software before it overflows, or else the system assumes an upset has occurred and activates RESET.

## PIN DESCRIPTION

### VCC

Main supply voltage (5V).

### VSS

Digital circuit ground (0V).

### VPD

RAM standby supply voltage (5V). This voltage must be present during normal operation. In a Power Down condition (i.e. VCC drops to zero), if  $\overline{\text{RESET}}$  is activated before VCC drops below spec and VPD continues to be held within spec, the top 16 bytes in the Register File will retain their contents.  $\overline{\text{RESET}}$  must be held low during the Power Down and should not be brought high until VCC is within spec and the oscillator has stabilized.

### VREF

Reference voltage for the A/D converter (5V). VREF is also the supply voltage to the analog portion of the A/D converter and the logic used to read Port 0.

### ANGND

Reference ground for the A/D converter. Should be held at nominally the same potential as VSS.

### VBB

Substrate voltage from the on-chip back-bias generator. This pin should be connected to ANGND through a 0.01  $\mu\text{f}$  capacitor (and not connected to anything else).

### XTAL1

Input of the oscillator inverter and of the internal clock generator.

### XTAL2

Output of the oscillator inverter.

### CLKOUT

Output of the internal clock generator. The frequency of CLKOUT is 1/3 the oscillator frequency. It has a 33% duty cycle.

## RESET

Reset input to the chip. Input low for at least 2 state times to reset the chip. The subsequent low-to-high transition re-synchronizes CLKOUT and commences a 10-state-time sequence in which the PSW is cleared and a jump to address 2080H is executed. Input high for normal operation.  $\overline{\text{RESET}}$  has an internal pullup.

## TEST

Input low enables a factory test mode. The user should tie this pin to VCC for normal operation.

## NMI

A positive transition clears the watchdog timer, and causes a vector to external memory location 0000H. External memory from 00H through 0FFH is reserved for Intel development systems.

## INST

Output high during an external memory read indicates the read is an instruction fetch.

## $\overline{\text{EA}}$

Input for memory select (External Access).  $\overline{\text{EA}} = 1$  causes memory accesses to locations 2000H through 3FFFH to be directed to on-chip ROM.  $\overline{\text{EA}} = 0$  causes accesses to these locations to be directed to off-chip memory.  $\overline{\text{EA}}$  has an internal pulldown, so it goes to 0 unless driven to 1.

## ALE

Address Latch Enable output. ALE is activated only during external memory accesses. It is used to latch the address from the multiplexed address/data bus.

## RD

Read signal output to external memory.  $\overline{\text{RD}}$  is activated only during external memory reads.

## WR

Write signal output to external memory.  $\overline{\text{WR}}$  is activated only during external memory writes.



**BHE**

Bus High Enable signal output to external memory. BHE = 0 selects the bank of memory that is connected to the high byte of the data bus. A0 = 0 selects the bank of memory that is connected to the low byte of the data bus. Thus accesses to a 16-bit wide memory can be to the low byte only (A0 = 0, BHE = 1), to the high byte only (A0 = 1, BHE = 0), or to both bytes (A0 = 0, BHE = 0). BHE is activated only when required during accesses to external memory. BHE can be ignored during read operations.

**READY**

The READY input is used to lengthen external memory bus cycles, for interfacing to slow or dynamic memory, or for bus sharing. If the pin is high CPU operation continues in a normal manner. If the pin is low prior to the rising edge of CLKOUT, the Memory Controller goes into a wait mode until the next negative transition in CLKOUT occurs with READY high. The bus cycle can be lengthened by up to 1  $\mu$ sec. When the external memory bus is not being used, READY has no effect. READY has a weak internal pullup, so it goes to 1 unless externally pulled low.

**HSI**

Inputs to High Speed Input Unit. Four HSI pins are available: HSI.0, HSI.1, HSI.2, and HSI.3. Two of them (HSI.2 and HSI.3) are shared with the HSO Unit.

**HSO**

Outputs from High Speed Output Unit. Six HSO pins are available: HSO.0, HSO.1, HSO.2, HSO.3, HSO.4, and HSO.5. Two of them (HSO.4 and HSO.5) are shared with the HSI Unit.

**Port 0**

8-bit high impedance input-only port. These pins can be used as digital inputs and/or as analog inputs to the on-chip A/D converter.

**Port 1**

8-bit quasi-bidirectional I/O port.

**Port 2**

8-bit multi-functional port. Six of its pins are shared with other functions in the 8096, the remaining 2 are quasi-bidirectional.

**Ports 3 and 4**

8-bit bi-directional I/O ports with open drain outputs. These pins are shared with the multiplexed address/data bus which has strong internal pullups.

**INSTRUCTION SET**

The 8096 instruction set makes use of six addressing modes as described below:

**DIRECT** — The operand is specified by an 8-bit address field in the instruction. The operand must be in the Register File or SFR space (locations 0000H through 00FFH).

**IMMEDIATE** — The operand itself follows the opcode in the instruction stream as immediate data. The immediate data can be either 8-bits or 16-bits as required by the opcode.

**INDIRECT** — An 8-bit address field in the instruction gives the address of a word register in the Register File which contains the 16-bit address of the operand. The operand can be anywhere in memory.

**INDIRECT WITH AUTO-INCREMENT** — Same as Indirect, except that, after the operand is referenced, the word register that contains the operand's address is incremented by 1 if the operand is a byte, or by 2 if the operand is a word.

**INDEXED** — The instruction contains an 8-bit address field and either an 8-bit or a 16-bit displacement field. The 8-bit address field gives the address of a word register in the Register File which contains a 16-bit base address. The 8- or 16-bit displacement field contains a signed displacement that will be added to the base address to produce the address of the operand. The operand can be anywhere in memory.

The 8096 contains a Zero Register at word address 0000H (and which contains 0000H). This register is available for performing comparisons and for use as a base register in indexed addressing. This effectively provides direct addressing to all 64K of memory.

In the 8096, the Stack Pointer is at word address 0018H in the Register File. If the 8-bit address field in an indexed instruction contains 18H, the Stack Pointer becomes the base register. This allows direct accessing of variables in the stack.

The following tables list the MCS-96 instructions, their opcodes, and execution times.

# Instruction Summary

| Mnemonic        | Operands | Operation (Note 1)   | Flags |   |   |   |    |    | Notes |
|-----------------|----------|--|-------|---|---|---|----|----|-------|
|                 |          |  | Z     | N | C | V | VT | ST |       |
| ADD/ADDB        | 2        | $D \leftarrow D + A$   | ✓     | ✓ | ✓ | ✓ | ↑  | —  |       |
| ADD/ADDB        | 3        | $D \leftarrow B + A$   | ✓     | ✓ | ✓ | ✓ | ↑  | —  |       |
| ADDC/ADDCB      | 2        | $D \leftarrow D + A + C$   | ↓     | ✓ | ✓ | ✓ | ↑  | —  |       |
| SUB/SUBB        | 2        | $D \leftarrow D - A$   | ✓     | ✓ | ✓ | ✓ | ↑  | —  |       |
| SUB/SUBB        | 3        | $D \leftarrow B - A$   | ✓     | ✓ | ✓ | ✓ | ↑  | —  |       |
| SUBC/SUBCB      | 2        | $D \leftarrow D - A + C - 1$   | ↓     | ✓ | ✓ | ✓ | ↑  | —  |       |
| CMP/CMPB        | 2        | $D - A$  | ✓     | ✓ | ✓ | ✓ | ↑  | —  |       |
| MUL/MULU        | 2        | $D, D + 2 \leftarrow D * A$  | —     | — | — | — | —  | ?  | 2     |
| MUL/MULU        | 3        | $D, D + 2 \leftarrow B * A$  | —     | — | — | — | —  | ?  | 2     |
| MULB/MULUB      | 2        | $D, D + 1 \leftarrow D * A$  | —     | — | — | — | —  | ?  | 3     |
| MULB/MULUB      | 3        | $D, D + 1 \leftarrow B * A$  | —     | — | — | — | —  | ?  | 3     |
| DIVU            | 2        | $D \leftarrow (D, D + 2)/A, D + 2 \leftarrow \text{remainder}$                           | —     | — | — | ✓ | ↑  | —  | 2     |
| DIVUB           | 2        | $D \leftarrow (D, D + 1)/A, D + 1 \leftarrow \text{remainder}$                           | —     | — | — | ✓ | ↑  | —  | 3     |
| DIV             | 2        | $D \leftarrow (D, D + 2)/A, D + 2 \leftarrow \text{remainder}$                           | —     | — | — | ? | ↑  | —  |       |
| DIVB            | 2        | $D \leftarrow (D, D + 1)/A, D + 1 \leftarrow \text{remainder}$                           | —     | — | — | ? | ↑  | —  |       |
| AND/ANDB        | 2        | $D \leftarrow D \text{ and } A$  | ✓     | ✓ | 0 | 0 | —  | —  |       |
| AND/ANDB        | 3        | $D \leftarrow B \text{ and } A$  | ✓     | ✓ | 0 | 0 | —  | —  |       |
| OR/ORB          | 2        | $D \leftarrow D \text{ or } A$   | ✓     | ✓ | 0 | 0 | —  | —  |       |
| XOR/XORB        | 2        | $D \leftarrow D \text{ (excl. or) } A$   | ✓     | ✓ | 0 | 0 | —  | —  |       |
| LD/LDB          | 2        | $D \leftarrow A$   | —     | — | — | — | —  | —  |       |
| ST/STB          | 2        | $A \leftarrow D$   | —     | — | — | — | —  | —  |       |
| LDBSE           | 2        | $D \leftarrow A; D + 1 \leftarrow \text{SIGN}(A)$  | —     | — | — | — | —  | —  | 3, 4  |
| LDBZE           | 2        | $D \leftarrow A; D + 1 \leftarrow 0$   | —     | — | — | — | —  | —  | 3, 4  |
| PUSH            | 1        | $SP \leftarrow SP - 2; (SP) \leftarrow A$  | —     | — | — | — | —  | —  |       |
| POP             | 1        | $A \leftarrow (SP); SP \leftarrow SP + 2$  | —     | — | — | — | —  | —  |       |
| PUSHF           | 0        | $SP \leftarrow SP - 2; (SP) \leftarrow PSW;$<br>$PSW \leftarrow 0000H$                   | 0     | 0 | 0 | 0 | 0  | 0  |       |
| POPF            | 0        | $PSW \leftarrow (SP); SP \leftarrow SP + 2;$<br>$I \leftarrow 0$                         | ✓     | ✓ | ✓ | ✓ | ✓  | ✓  |       |
| SJMP            | 1        | $PC \leftarrow PC + 11\text{-bit offset}$  | —     | — | — | — | —  | —  | 5     |
| LJMP            | 1        | $PC \leftarrow PC + 16\text{-bit offset}$  | —     | — | — | — | —  | —  | 5     |
| BR [indirect]   | 1        | $PC \leftarrow (A)$  | —     | — | — | — | —  | —  |       |
| SCALL           | 1        | $SP \leftarrow SP - 2; (SP) \leftarrow PC;$<br>$PC \leftarrow PC + 11\text{-bit offset}$ | —     | — | — | — | —  | —  | 5     |
| LCALL           | 1        | $SP \leftarrow SP - 2; (SP) \leftarrow PC;$<br>$PC \leftarrow PC + 16\text{-bit offset}$ | —     | — | — | — | —  | —  | 5     |
| RET             | 0        | $PC \leftarrow (SP); SP \leftarrow SP + 2$   | —     | — | — | — | —  | —  |       |
| J (conditional) | 1        | $PC \leftarrow PC + 8\text{-bit offset (if taken)}$                                      | —     | — | — | — | —  | —  | 5     |
| JC              | 1        | Jump if C = 1  | —     | — | — | — | —  | —  | 5     |
| JNC             | 1        | Jump if C = 0  | —     | — | — | — | —  | —  | 5     |
| JE              | 1        | Jump if Z = 1  | —     | — | — | — | —  | —  | 5     |

## NOTES:

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
2. D, D + 2 are consecutive WORDS in memory; D is DOUBLE-WORD aligned.
3. D, D + 1 are consecutive BYTES in memory; D is WORD aligned.
4. Changes a byte to a word.
5. Offset is a 2's complement number.

Instruction Summary (Continued)

| Mnemonic         | Operands | Operation (Note 1)                                 | Flags |   |   |   |    |    | Notes |
|------------------|----------|--|-------|---|---|---|----|----|-------|
|                  |          |  | Z     | N | C | V | VT | ST |       |
| JNE              | 1        | Jump if Z = 0                                      | —     | — | — | — | —  | —  | 5     |
| JGE              | 1        | Jump if N = 0                                      | —     | — | — | — | —  | —  | 5     |
| JLT              | 1        | Jump if N = 1                                      | —     | — | — | — | —  | —  | 5     |
| JGT              | 1        | Jump if N = 0 and Z = 0                            | —     | — | — | — | —  | —  | 5     |
| JLE              | 1        | Jump if N = 1 or Z = 1                             | —     | — | — | — | —  | —  | 5     |
| JH               | 1        | Jump if C = 1 and Z = 0                            | —     | — | — | — | —  | —  | 5     |
| JNH              | 1        | Jump if C = 0 or Z = 1                             | —     | — | — | — | —  | —  | 5     |
| JV               | 1        | Jump if V = 1                                      | —     | — | — | — | —  | —  | 5     |
| JNV              | 1        | Jump if V = 0                                      | —     | — | — | — | —  | —  | 5     |
| JVT              | 1        | Jump if VT = 1; Clear VT                           | —     | — | — | — | 0  | —  | 5     |
| JNVT             | 1        | Jump if VT = 0; Clear VT                           | —     | — | — | — | 0  | —  | 5     |
| JST              | 1        | Jump if ST = 1                                     | —     | — | — | — | —  | —  | 5     |
| JNST             | 1        | Jump if ST = 0                                     | —     | — | — | — | —  | —  | 5     |
| JBS              | 3        | Jump if Specified Bit = 1                          | —     | — | — | — | —  | —  | 5, 6  |
| JBC              | 3        | Jump if Specified Bit = 0                          | —     | — | — | — | —  | —  | 5, 6  |
| DJNZ             | 1        | D ← D - 1; if D ≠ 0 then<br>PC ← PC + 8-bit offset | —     | — | — | — | —  | —  | 5     |
| DEC/DECB         | 1        | D ← D - 1  | ✓     | ✓ | ✓ | ✓ | ↑  | —  |       |
| NEG/NEGB         | 1        | D ← 0 - D  | ✓     | ✓ | ✓ | ✓ | ↑  | —  |       |
| INC/INCB         | 1        | D ← D + 1  | ✓     | ✓ | ✓ | ✓ | ↑  | —  |       |
| EXT              | 1        | D ← D; D + 2 ← Sign (D)                            | ✓     | ✓ | 0 | 0 | —  | —  | 2     |
| EXTB             | 1        | D ← D; D + 1 ← Sign(D)                             | ✓     | ✓ | 0 | 0 | —  | —  | 3     |
| NOT/NOTB         | 1        | D ← Logical Not (D)                                | ✓     | ✓ | 0 | 0 | —  | —  |       |
| CLR/CLRB         | 1        | D ← 0  | 1     | 0 | 0 | 0 | —  | —  |       |
| SHL/SHLB/SHLL    | 2        | C ← msb ———— lsb ← 0                               | ✓     | ? | ✓ | ✓ | ↑  | —  | 7     |
| SHR/SHRB/SHRL    | 2        | 0 → msb ———— lsb → C                               | ✓     | ? | ✓ | 0 | —  | ✓  | 7     |
| SHRA/SHRAB/SHRAL | 2        | msb → msb ———— lsb → C                             | ✓     | ✓ | ✓ | 0 | —  | ✓  | 7     |
| SETC             | 0        | C ← 1  | —     | — | 1 | — | —  | —  |       |
| CLRC             | 0        | C ← 0  | —     | — | 0 | — | —  | —  |       |
| CLRVT            | 0        | VT ← 0   | —     | — | — | — | 0  | —  |       |
| RST              | 0        | PC ← 2080H   | 0     | 0 | 0 | 0 | 0  | 0  | 8     |
| DI               | 0        | Disable All Interrupts (I ← 0)                     | —     | — | — | — | —  | —  |       |
| EI               | 0        | Enable All Interrupts (I ← 1)                      | —     | — | — | — | —  | —  |       |
| NOP              | 0        | PC ← PC + 1  | —     | — | — | — | —  | —  |       |
| SKIP             | 0        | PC ← PC + 2  | —     | — | — | — | —  | —  |       |
| NORML            | 2        | Left shift till msb = 1; D ← shift count           | ✓     | ? | 0 | — | —  | —  | 7     |
| TRAP             | 0        | SP ← SP - 2; (SP) ← PC<br>PC ← (2010H)             | —     | — | — | — | —  | —  | 9     |

NOTES:

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
5. Offset is a 2's complement number.
6. Specified bit is one of the 2048 bits in the register file.
7. The "L" (Long) suffix indicates double-word operation.
8. Initiates a Reset by pulling RESET low. Software should re-initialize all the necessary registers with code starting at 2080H.
9. The assembler will not accept this mnemonic.

| MNEMONIC                | OPERANDS | DIRECT |       |                | IMMEDIATE |       |                | INDIRECT® |       |                 |           |                 | INDEXED® |       |                 |       |                  |
|-------------------------|----------|--------|-------|----------------|-----------|-------|----------------|-----------|-------|-----------------|-----------|-----------------|----------|-------|-----------------|-------|------------------|
|                         |          | OPCODE | BYTES | STATE<br>TIMES | OPCODE    | BYTES | STATE<br>TIMES | NORMAL    |       |                 | AUTO-INC. |                 | SHORT    |       |                 | LONG  |                  |
|                         |          |        |       |                |           |       |                | OPCODE    | BYTES | STATE®<br>TIMES | BYTES     | STATE®<br>TIMES | OPCODE   | BYTES | STATE®<br>TIMES | BYTES | STATE®<br>TIMES® |
|                         |          |        |       |                |           |       |                |           |       |                 |           |                 |          |       |                 |       |                  |
| ARITHMETIC INSTRUCTIONS |          |        |       |                |           |       |                |           |       |                 |           |                 |          |       |                 |       |                  |
| ADD                     | 2        | 64     | 3     | 4              | 65        | 4     | 5              | 66        | 3     | 6/11            | 3         | 7/12            | 67       | 4     | 6/11            | 5     | 7/12             |
| ADD                     | 3        | 44     | 4     | 5              | 45        | 5     | 6              | 46        | 4     | 7/12            | 4         | 8/13            | 47       | 5     | 7/12            | 6     | 8/13             |
| ADDB                    | 2        | 74     | 3     | 4              | 75        | 3     | 4              | 76        | 3     | 6/11            | 3         | 7/12            | 77       | 4     | 6/11            | 5     | 7/12             |
| ADDB                    | 3        | 54     | 4     | 5              | 55        | 4     | 5              | 56        | 4     | 7/12            | 4         | 8/13            | 57       | 5     | 7/12            | 6     | 8/13             |
| ADDC                    | 2        | A4     | 3     | 4              | A5        | 4     | 5              | A6        | 3     | 6/11            | 3         | 7/12            | A7       | 4     | 6/11            | 5     | 7/12             |
| ADDCB                   | 2        | B4     | 3     | 4              | B5        | 3     | 4              | B6        | 3     | 6/11            | 3         | 7/12            | B7       | 4     | 6/11            | 5     | 7/12             |
| SUB                     | 2        | 68     | 3     | 4              | 69        | 4     | 5              | 6A        | 3     | 6/11            | 3         | 7/12            | 6B       | 4     | 6/11            | 5     | 7/12             |
| SUB                     | 3        | 48     | 4     | 5              | 49        | 5     | 6              | 4A        | 4     | 7/12            | 4         | 8/13            | 4B       | 5     | 7/12            | 6     | 8/13             |
| SUBB                    | 2        | 78     | 3     | 4              | 79        | 3     | 4              | 7A        | 3     | 6/11            | 3         | 7/12            | 7B       | 4     | 6/11            | 5     | 7/12             |
| SUBB                    | 3        | 58     | 4     | 5              | 59        | 4     | 5              | 5A        | 4     | 7/12            | 4         | 8/13            | 5B       | 5     | 7/12            | 6     | 8/13             |
| SUBC                    | 2        | A8     | 3     | 4              | A9        | 4     | 5              | AA        | 3     | 6/11            | 3         | 7/12            | AB       | 4     | 6/11            | 5     | 7/12             |
| SUBCB                   | 2        | B8     | 3     | 4              | B9        | 3     | 4              | BA        | 3     | 6/11            | 3         | 7/12            | BB       | 4     | 6/11            | 5     | 7/12             |
| CMP                     | 2        | 88     | 3     | 4              | 89        | 4     | 5              | 8A        | 3     | 6/11            | 3         | 7/12            | 8B       | 4     | 6/11            | 5     | 7/12             |
| CMPB                    | 2        | 98     | 3     | 4              | 99        | 3     | 4              | 9A        | 3     | 6/11            | 3         | 7/12            | 9B       | 4     | 6/11            | 5     | 7/12             |
|                         |          |        |       |                |           |       |                |           |       |                 |           |                 |          |       |                 |       |                  |
| MULU                    | 2        | 6C     | 3     | 25             | 6D        | 4     | 26             | 6E        | 3     | 27/32           | 3         | 28/33           | 6F       | 4     | 27/32           | 5     | 28/33            |
| MULU                    | 3        | 4C     | 4     | 26             | 4D        | 5     | 27             | 4E        | 4     | 28/33           | 4         | 29/34           | 4F       | 5     | 28/33           | 6     | 29/34            |
| MULUB                   | 2        | 7C     | 3     | 17             | 7D        | 3     | 17             | 7E        | 3     | 19/24           | 3         | 20/25           | 7F       | 4     | 19/24           | 5     | 20/25            |
| MULUB                   | 3        | 5C     | 4     | 18             | 5D        | 4     | 18             | 5E        | 4     | 20/25           | 4         | 21/26           | 5F       | 5     | 20/25           | 6     | 21/26            |
| MUL                     | 2        | ②      | 4     | 29             | ②         | 5     | 30             | ②         | 4     | 31/36           | 4         | 32/37           | ②        | 5     | 31/36           | 6     | 32/37            |
| MUL                     | 3        | ②      | 5     | 30             | ②         | 6     | 31             | ②         | 5     | 32/37           | 5         | 33/38           | ②        | 6     | 32/37           | 7     | 33/38            |
| MULB                    | 2        | ②      | 4     | 21             | ②         | 4     | 21             | ②         | 4     | 23/28           | 4         | 24/29           | ②        | 5     | 23/28           | 6     | 24/29            |
| MULB                    | 3        | ②      | 5     | 22             | ②         | 5     | 22             | ②         | 5     | 24/29           | 5         | 25/30           | ②        | 6     | 24/29           | 7     | 25/30            |
| DIVU                    | 2        | 8C     | 3     | 25             | 8D        | 4     | 26             | 8E        | 3     | 28/32           | 3         | 29/33           | 8F       | 4     | 28/32           | 5     | 29/33            |
| DIVUB                   | 2        | 9C     | 3     | 17             | 9D        | 3     | 17             | 9E        | 3     | 20/24           | 3         | 21/25           | 9F       | 4     | 20/24           | 5     | 21/25            |
| DIV                     | 2        | ②      | 4     | 29             | ②         | 5     | 30             | ②         | 4     | 32/36           | 4         | 33/37           | ②        | 5     | 32/36           | 6     | 33/37            |
| DIVB                    | 2        | ②      | 4     | 21             | ②         | 4     | 21             | ②         | 4     | 24/28           | 4         | 25/29           | ②        | 5     | 24/28           | 6     | 25/29            |

**Notes:**

- ② Long indexed and Indirect+ instructions have identical opcodes with Short indexed and Indirect modes, respectively. The second byte of instructions using any indirect or indexed addressing mode specifies the exact mode used. If the second byte is even, use Indirect or Short Indexed. If it is odd, use Indirect+ or Long indexed. In all cases the second byte of the instruction always specifies an even (word) location for the address referenced.
- ① Number of state times shown for internal/external operands.
- ② The opcodes for signed multiply and divide are the opcodes for the unsigned functions with an "FE" appended as a prefix.



| MNEMONIC                          | OPERANDS           | DIRECT |       |             | IMMEDIATE |       |             | INDIRECT®          |       |                          |       | INDEXED®                 |        |       |                          |       |                          |
|-----------------------------------|--------------------|--------|-------|-------------|-----------|-------|-------------|--------------------|-------|--------------------------|-------|--------------------------|--------|-------|--------------------------|-------|--------------------------|
|                                   |                    | OPCODE | BYTES | STATE TIMES | OPCODE    | BYTES | STATE TIMES | NORMAL             |       | AUTO-INC.                |       | SHORT                    |        | LONG  |                          |       |                          |
|                                   |                    |        |       |             |           |       |             | OPCODE             | BYTES | STATE <sup>①</sup> TIMES | BYTES | STATE <sup>①</sup> TIMES | OPCODE | BYTES | STATE <sup>①</sup> TIMES | BYTES | STATE <sup>①</sup> TIMES |
| LOGICAL INSTRUCTIONS              |                    |        |       |             |           |       |             |                    |       |                          |       |                          |        |       |                          |       |                          |
| AND                               | 2                  | 60     | 3     | 4           | 61        | 4     | 5           | 62                 | 3     | 6/11                     | 3     | 7/12                     | 63     | 4     | 6/11                     | 5     | 7/12                     |
| AND                               | 3                  | 40     | 4     | 5           | 41        | 5     | 6           | 42                 | 4     | 7/12                     | 4     | 8/13                     | 43     | 5     | 7/12                     | 6     | 8/13                     |
| ANDB                              | 2                  | 70     | 3     | 4           | 71        | 3     | 4           | 72                 | 3     | 6/11                     | 3     | 7/12                     | 73     | 4     | 6/11                     | 5     | 7/12                     |
| ANDB                              | 3                  | 50     | 4     | 5           | 51        | 4     | 5           | 52                 | 4     | 7/12                     | 4     | 8/13                     | 53     | 5     | 7/12                     | 6     | 8/13                     |
| OR                                | 2                  | 80     | 3     | 4           | 81        | 4     | 5           | 82                 | 3     | 6/11                     | 3     | 7/12                     | 83     | 4     | 6/11                     | 5     | 7/12                     |
| ORB                               | 2                  | 90     | 3     | 4           | 91        | 3     | 4           | 92                 | 3     | 6/11                     | 3     | 7/12                     | 93     | 4     | 6/11                     | 5     | 7/12                     |
| XOR                               | 2                  | 84     | 3     | 4           | 85        | 4     | 5           | 86                 | 3     | 6/11                     | 3     | 7/12                     | 87     | 4     | 6/11                     | 5     | 7/12                     |
| XORB                              | 2                  | 94     | 3     | 4           | 95        | 3     | 4           | 96                 | 3     | 6/11                     | 3     | 7/12                     | 97     | 4     | 6/11                     | 5     | 7/12                     |
| DATA TRANSFER INSTRUCTIONS        |                    |        |       |             |           |       |             |                    |       |                          |       |                          |        |       |                          |       |                          |
| LD                                | 2                  | A0     | 3     | 4           | A1        | 4     | 5           | A2                 | 3     | 6/11                     | 3     | 7/12                     | A3     | 4     | 6/11                     | 5     | 7/12                     |
| LDB                               | 2                  | B0     | 3     | 4           | B1        | 3     | 4           | B2                 | 3     | 6/11                     | 3     | 7/12                     | B3     | 4     | 6/11                     | 5     | 7/12                     |
| ST                                | 2                  | C0     | 3     | 4           | —         | —     | —           | C2                 | 3     | 7/11                     | 3     | 8/12                     | C3     | 4     | 7/11                     | 5     | 8/12                     |
| STB                               | 2                  | C4     | 3     | 4           | —         | —     | —           | C6                 | 3     | 7/11                     | 3     | 8/12                     | C7     | 4     | 7/11                     | 5     | 8/12                     |
| LDBSE                             | 2                  | BC     | 3     | 4           | BD        | 3     | 4           | BE                 | 3     | 6/11                     | 3     | 7/12                     | BF     | 4     | 6/11                     | 5     | 7/12                     |
| LDBZE                             | 2                  | AC     | 3     | 4           | AD        | 3     | 4           | AE                 | 3     | 6/11                     | 3     | 7/12                     | AF     | 4     | 6/11                     | 5     | 7/12                     |
| STACK OPERATIONS (internal stack) |                    |        |       |             |           |       |             |                    |       |                          |       |                          |        |       |                          |       |                          |
| PUSH                              | 1                  | C8     | 2     | 8           | C9        | 3     | 8           | CA                 | 2     | 11/15                    | 2     | 12/16                    | CB     | 3     | 11/15                    | 4     | 12/16                    |
| POP                               | 1                  | CC     | 2     | 12          | —         | —     | —           | CE                 | 2     | 14/18                    | 2     | 14/18                    | CF     | 3     | 14/18                    | 4     | 14/18                    |
| PUSHF                             | 0                  | F2     | 1     | 8           |           |       |             |                    |       |                          |       |                          |        |       |                          |       |                          |
| POPF                              | 0                  | F3     | 1     | 9           |           |       |             |                    |       |                          |       |                          |        |       |                          |       |                          |
| STACK OPERATIONS (external stack) |                    |        |       |             |           |       |             |                    |       |                          |       |                          |        |       |                          |       |                          |
| PUSH                              | 1                  | C8     | 2     | 12          | C9        | 3     | 12          | CA                 | 2     | 15/19                    | 2     | 16/20                    | CB     | 3     | 15/19                    | 4     | 16/20                    |
| POP                               | 1                  | CC     | 2     | 14          | —         | —     | —           | CE                 | 2     | 16/20                    | 2     | 16/20                    | CF     | 3     | 16/20                    | 4     | 16/20                    |
| PUSHF                             | 0                  | F2     | 1     | 12          |           |       |             |                    |       |                          |       |                          |        |       |                          |       |                          |
| POPF                              | 0                  | F3     | 1     | 13          |           |       |             |                    |       |                          |       |                          |        |       |                          |       |                          |
| JUMPS AND CALLS                   |                    |        |       |             |           |       |             |                    |       |                          |       |                          |        |       |                          |       |                          |
| MNEMONIC                          | OPCODE             |        | BYTES |             | STATES    |       | MNEMONIC    | OPCODE             |       | BYTES                    |       | STATES                   |        |       |                          |       |                          |
| LJMP                              | E7                 |        | 3     |             | 8         |       | LCALL       | EF                 |       | 3                        |       | 13/16 <sup>⑤</sup>       |        |       |                          |       |                          |
| SJMP                              | 20-27 <sup>④</sup> |        | 2     |             | 8         |       | SCALL       | 28-2F <sup>④</sup> |       | 2                        |       | 13/16 <sup>⑤</sup>       |        |       |                          |       |                          |
| BR[ ]                             | E3                 |        | 2     |             | 8         |       | RET         | F0                 |       | 1                        |       | 12/16 <sup>⑤</sup>       |        |       |                          |       |                          |
| Notes:                            |                    |        |       |             |           |       |             | TRAP <sup>③</sup>  |       | F7                       |       | 1                        |        | 21/24 |                          |       |                          |

## Notes:

- ① Number of state times shown for internal/external operands.
- ③ The assembler does not accept this mnemonic.
- ④ The least significant 3 bits of the opcode are concatenated with the following 8 bits to form an 11-bit, 2's complement, offset for the relative call or jump.
- ⑤ State times for stack located internal/external.

### CONDITIONAL JUMPS

| All conditional jumps are 2 byte instructions. They require 8 state times if the jump is taken, 4 if it is not. |        |          |        |          |        |          |        |
|---|--------|----------|--------|----------|--------|----------|--------|
| MNEMONIC  | OPCODE | MNEMONIC | OPCODE | MNEMONIC | OPCODE | MNEMONIC | OPCODE |
| JC  | DB     | JE       | DF     | JGE      | D6     | JGT      | D2     |
| JNC   | D3     | JNE      | D7     | JLT      | DE     | JLE      | DA     |
| JH  | D9     | JV       | DD     | JVT      | DC     | JST      | D8     |
| JNH   | D1     | JNV      | D5     | JNVT     | D4     | JNST     | D0     |

### JUMP ON BIT CLEAR OR BIT SET

| These instructions are 3-byte instructions. They require 9 state times if the jump is taken, 5 if it is not. |            |    |    |    |    |    |    |    |
|--|------------|----|----|----|----|----|----|----|
| MNEMONIC   | BIT NUMBER |    |    |    |    |    |    |    |
|  | 0          | 1  | 2  | 3  | 4  | 5  | 6  | 7  |
| JBC  | 30         | 31 | 32 | 33 | 34 | 35 | 36 | 37 |
| JBS  | 38         | 39 | 3A | 3B | 3C | 3D | 3E | 3F |

### LOOP CONTROL

|      |            |          |                                   |
|------|------------|----------|-----------------------------------|
| DJNZ | OPCODE EO; | 3 BYTES; | 5/9 STATE TIMES (NOT TAKEN/TAKEN) |
|------|------------|----------|-----------------------------------|

### SINGLE REGISTER INSTRUCTIONS

| MNEMONIC | OPCODE | BYTES | STATES | MNEMONIC | OPCODE | BYTES | STATES |
|----------|--------|-------|--------|----------|--------|-------|--------|
| DEC      | 05     | 2     | 4      | EXT      | 06     | 2     | 4      |
| DECB     | 15     | 2     | 4      | EXTB     | 16     | 2     | 4      |
| NEG      | 03     | 2     | 4      | NOT      | 02     | 2     | 4      |
| NEGB     | 13     | 2     | 4      | NOTB     | 12     | 2     | 4      |
| INC      | 07     | 2     | 4      | CLR      | 01     | 2     | 4      |
| INCB     | 17     | 2     | 4      | CLRB     | 11     | 2     | 4      |

### SHIFT INSTRUCTIONS

| INSTR<br>MNEMONIC | WORD |   | INSTR<br>MNEMONIC | BYTE |   | INSTR<br>MNEMONIC | DBL WD |   | STATE TIMES                  |
|-------------------|------|---|-------------------|------|---|-------------------|--------|---|------------------------------|
|                   | OP   | B |                   | OP   | B |                   | OP     | B |                              |
| SHL               | 09   | 3 | SHLB              | 19   | 3 | SHLL              | 0D     | 3 | 7 + 1 PER SHIFT <sup>Ⓞ</sup> |
| SHR               | 08   | 3 | SHRB              | 18   | 3 | SHRL              | 0C     | 3 | 7 + 1 PER SHIFT <sup>Ⓞ</sup> |
| SHRA              | 0A   | 3 | SHRAB             | 1A   | 3 | SHRAL             | 0E     | 3 | 7 + 1 PER SHIFT <sup>Ⓞ</sup> |

### SPECIAL CONTROL INSTRUCTIONS

| MNEMONIC | OPCODE | BYTES | STATES | MNEMONIC | OPCODE | BYTES | STATES |
|----------|--------|-------|--------|----------|--------|-------|--------|
| SETC     | F9     | 1     | 4      | DI       | FA     | 1     | 4      |
| CLRC     | F8     | 1     | 4      | EI       | FB     | 1     | 4      |
| CLRVT    | FC     | 1     | 4      | NOP      | FD     | 1     | 4      |
| RST      | FF     | 1     | 166    | SKIP     | 00     | 2     | 4      |

### NORMALIZE

|       |    |   |                  |
|-------|----|---|------------------|
| NORML | 0F | 3 | 11 + 1 PER SHIFT |
|-------|----|---|------------------|

#### Notes:

- Ⓞ This instruction takes 2 states to pull RST low, then holds it low for 2 states to initiate a reset. The reset takes 12 states, at which time the program restarts at location 2080H.
- Ⓞ Execution will take at least 8 states, even for 0 shift.

## FUNCTIONAL DEVIATIONS

Functional deviations from the 809x and 839x on the 809x-90 and 839x-90.

### CPU SECTION

1. Indexed, 3 Word Multiply — The displacement portion of an indexed, three word multiply may not be in the range of 200H thru 17FFH inclusive.
2. Add or Subtract with carry — The zero flag is both set and cleared by these instructions.
3. EXT — This instruction never sets the N flag, and always sets the Z flag. The EXTB works correctly.
4. Read-Modify-Write on Interrupt Pending — A read-modify-write instruction on the interrupt pending register may cause interrupts that occur during execution of the instruction to be missed.
5. READY line — The READY line should not be held active during the execution of an instruction that accesses HSI\_TIME, SP\_STAT or IOS1. It should also not be active for a data write during the instruction immediately preceding one of the above operations.
6. Signed Divide — The V and VT flags may indicate an overflow after a signed divide when no overflow has occurred.

### HSI/HSO SECTION

1. HSI Timing — An event occurring within 16 state times of a prior event on the same HSI line may not be recorded. Additionally, an event occurring within 16 state times of a prior event on another HSI line may be recorded with a time tag one count earlier than expected. Events are defined as the condition the line is set to trigger on.

2. HSI Divide by 8 Mode — If an event on a pin set to look for every eighth transition occurs less than 16 state times after an event on any other pin, then the divide by 8 event will be recorded twice in the HSI FIFO. The time tag of the duplicate FIFO entry will be equal to that of the initial entry plus one.
3. HSO Interrupts — Software timer interrupts cannot be generated by the HSO commands that reset Timer 2 or start an A to D conversion.

### SERIAL PORT SECTION

1. Serial Port Flags — Reading SP\_STAT may not clear the TI or RI flag if that flag was set within two state times prior to the read.
2. Serial Port Mode 0 — The serial port is not tested in mode 0. The receive function in this mode does not work correctly.
3. Serial Port Baud Value — Loading the baud rate register with 8000H (maximum baud rate, internal clock) may cause an 11 millisecond delay (at  $F_{osc} = 12 \text{ Mhz}$ ) before the port is properly initialized. After initialization the port works properly.

### STANDARD I/O SECTION

1. Ports 3 and 4 (Internal Execution Mode Only) — To be used as outputs, Ports 3 and 4 each must be addressed as words but written to as bytes. To write to Port 3 use "ST temp,1ffh", where the low byte of "temp" contains the data for the port. To write to Port 4, use the DCB operator to generate the opcode sequence "0C3H, 001H, 0FFH, 01FH, (temp)", where the high byte of "temp" contains the data for the port. Ports 3 and 4 will not work as input ports.

## ADDITIONAL INFORMATION

The following information was not in the 1984 "8096 Users Manual"

1. After a chip reset the watchdog timer will not run until a "01EH" followed by a "0E1H" is written to the watchdog timer register. After this is done the watchdog timer functions as described in the users manual until the next chip reset. This feature permits disabling of the watchdog by simply not writing to it.

2. External interrupts on P0.7 are sampled every state time instead of every eight state times.

3. The baud rate generated in the external clock mode is four times faster than stated in the Users Manual. The correct formula for other than mode 0 using T2CLK as the input frequency is:

$$\text{Baud Rate} = \text{Input Frequency} / (16 \cdot B).$$

4. If more than one HSO event is scheduled to occur with interrupts at the same time multiple HSO interrupts may occur. This is because HSO inter-

rupts are internal events and as such are not synchronized to Timer1.

5. Locations 2012H through 207FH in external memory must be filled with the hex value 0FFFFH to ensure compatibility with future parts. The internal locations in this range are still reserved for the factory test code.

6. There are several restrictions on using the special function registers.

A. Neither the source nor the destination addresses of the Multiply and Divide instructions can be a writable special function register.

B. These registers may not be used as base or index registers for indexed or indirect instructions.

C. Several of these registers can only be accessed as words, while others only as bytes. These restrictions are listed in Chapter 3 of the manual.

6. Signed Divide — The V and CY flags may indicate an overflow after a signed divide when no overflow has occurred.

## HSO SECTION

1. HSI Timing — An event occurring within 18 state times of a chip event on the same HSI line may not be recorded. Additionally, an event occurring within 18 state times of a chip event on another HSI line may be recorded with a time lag one count earlier than expected. Events are defined as the condition the line is set to input or



# **ELECTRICAL CHARACTERISTICS** **ABSOLUTE MAXIMUM RATINGS\***

Ambient Temperature Under Bias . . . 0°C to +70°C  
Storage Temperature . . . . . -40°C to +150°C  
Voltage from Any Pin to  
VSS or ANGND . . . . . -0.3V to +7.0V  
Average Output Current from Any Pin . . . . . 10 mA  
Power Dissipation . . . . . 1.5 Watts

\*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

## **OPERATING CONDITIONS**

| Symbol | Parameter                      | Min       | Max       | Units |
|--------|--------------------------------|-----------|-----------|-------|
| TA     | Ambient Temperature Under Bias | 0         | +70       | C     |
| VCC    | Digital Supply Voltage         | 4.50      | 5.50      | V     |
| VREF   | Analog Supply Voltage          | 4.5       | 5.5       | V     |
|        |                                | VCC - 0.3 | VCC + 0.3 | V     |
| fOSC   | Oscillator Frequency           | 6.0       | 12        | MHz   |
| VPD    | Power-Down Supply Voltage      | 4.50      | 5.50      | V     |

VBB should be connected to ANGND through a 0.01 µF capacitor. ANGND and VSS should be nominally at the same potential.

## **DC CHARACTERISTICS**

| Symbol | Parameter  | Min  | Max       | Units | Test Conditions                  |
|--------|--|------|-----------|-------|----------------------------------|
| VIL    | Input Low Voltage (Except RESET)                                   | -0.3 | +0.8      | V     |                                  |
| VIL1   | Input Low Voltage, RESET   | -0.3 | +0.7      | V     |                                  |
| VIH    | Input High Voltage (Except RESET, NMI, XTAL1)                      | 2.0  | VCC + 0.5 | V     |                                  |
| VIH1   | Input High Voltage, RESET Rising                                   | 2.4  | VCC + 0.5 | V     |                                  |
| VIH2   | Input High Voltage, RESET Falling                                  | 2.1  | VCC + 0.5 | V     |                                  |
| VIH3   | Input High Voltage, NMI, XTAL1                                     | 2.4  | VCC + 0.5 | V     |                                  |
| VOL    | Output Low Voltage   |      | 0.45      | V     | See Note 1.                      |
| VOH    | Output High Voltage  | 2.4  |           | V     | See Note 2.                      |
| ICC    | VCC Supply Current   |      | 200       | mA    | All outputs disconnected.        |
| IPD    | VPD Supply Current   |      | 1         | mA    | Normal operation and Power-Down. |
| IREF   | VREF Supply Current  |      | 8         | mA    |                                  |
| ILI    | Input Leakage Current to all pins of HSI, P0, P3, P4, and to P2.1. |      | ± 10      | µA    | Vin = 0 to VCC<br>See Note 3     |
| IIH    | Input High Current to EA   |      | 100       | µA    | VIH = 2.4V                       |
| IIL    | Input Low Current to all pins of P1, and to P2.6, P2.7.            |      | -100      | µA    | VIL = 0.45V                      |
| IIL1   | Input Low Current to RESET   |      | -2        | mA    | VIL = 0.45V                      |
| IIL2   | Input Low Current P2.2, P2.3, P2.4, READY                          |      | -50       | µA    | VIL = 0.45V                      |
| Cs     | Pin Capacitance (Any Pin to VSS)                                   |      | 10        | pF    | fTEST = 1 MHz                    |

### **NOTES:**

1. IOL = 0.36 mA for all pins of P1, for P2.6 and P2.7, and for all pins of P3 and P4 when used as ports.  
IOL = 2.0 mA for TXD, RXD (in serial port mode 0), PWM, CLKOUT, ALE, BHE, RD, WR, and all pins of HSO and P3 and P4 when used as external memory bus (AD0-AD15).
2. IOH = -20 µA for all pins of P1, for P2.6 and P2.7.  
IOH = -200 µA for TXD, RXD (in serial port mode 0), PWM, CLKOUT, ALE, BHE, WR, and all pins of HSO and P3 and P4 when used as external memory bus (AD0-AD15).  
P3 and P4, when used as ports, have open-drain outputs.
3. Analog Conversion not in process.

**A/D CONVERTER SPECIFICATIONS**

A/D Converter operation is verified only on the 8097, 8397, 8095, 8395.

The absolute conversion accuracy is dependent on the accuracy of VREF. The specifications given below assume adherence to the Operating Conditions section of these data sheets. Testing is done at VREF = 5.120 volts.

|                             |                      |
|-----------------------------|----------------------|
| Resolution                  | $\pm 0.001$ VREF     |
| Accuracy                    | $\pm 0.004$ VREF     |
| Differential nonlinearity   | $\pm 0.002$ VREF max |
| Integral nonlinearity       | $\pm 0.004$ VREF max |
| Channel-to-channel matching | $\pm 1$ LSB          |
| Crosstalk (DC to 100 kHz)   | -60 dB max           |

**AC CHARACTERISTICS** (VCC, VPD = 4.5 to 5.5 Volts; T<sub>A</sub> = 0°C to 70°C; fosc = 6.0 to 12.0 MHz)

Test Conditions: Load capacitance on output pins = 80 pF

Oscillator Frequency = 12.00 MHz

**Timing Requirements** (other system components must meet these specs)

| Symbol | Parameter                            | Min                     | Max                        | Units |
|--------|--------------------------------------|-------------------------|----------------------------|-------|
| TCLYX  | READY Hold after CLKOUT falling edge | 0 (1)                   |                            | nsec  |
| TLLYV  | End of ALE to READY Setup            | - T <sub>osc</sub>      | 2T <sub>osc</sub> - 60     | nsec  |
| TLLYH  | End of ALE to READY high             | 2 T <sub>osc</sub> + 60 | 4T <sub>osc</sub> - 60 (2) | nsec  |
| TYLYH  | Non-ready time                       |                         | 1000                       | nsec  |
| TAVDV  | Address Valid to Input Data Valid    |                         | 5T <sub>osc</sub> - 90     | nsec  |
| TRLDV  | RD/Active to Input Data Valid        |                         | 3T <sub>osc</sub> - 60     | nsec  |
| TRXDX  | Data Hold after RD/inactive (3)      | 0                       |                            | nsec  |
| TRXDZ  | RD/Inactive to Input Data Float (3)  |                         | T <sub>osc</sub> - 20      | nsec  |

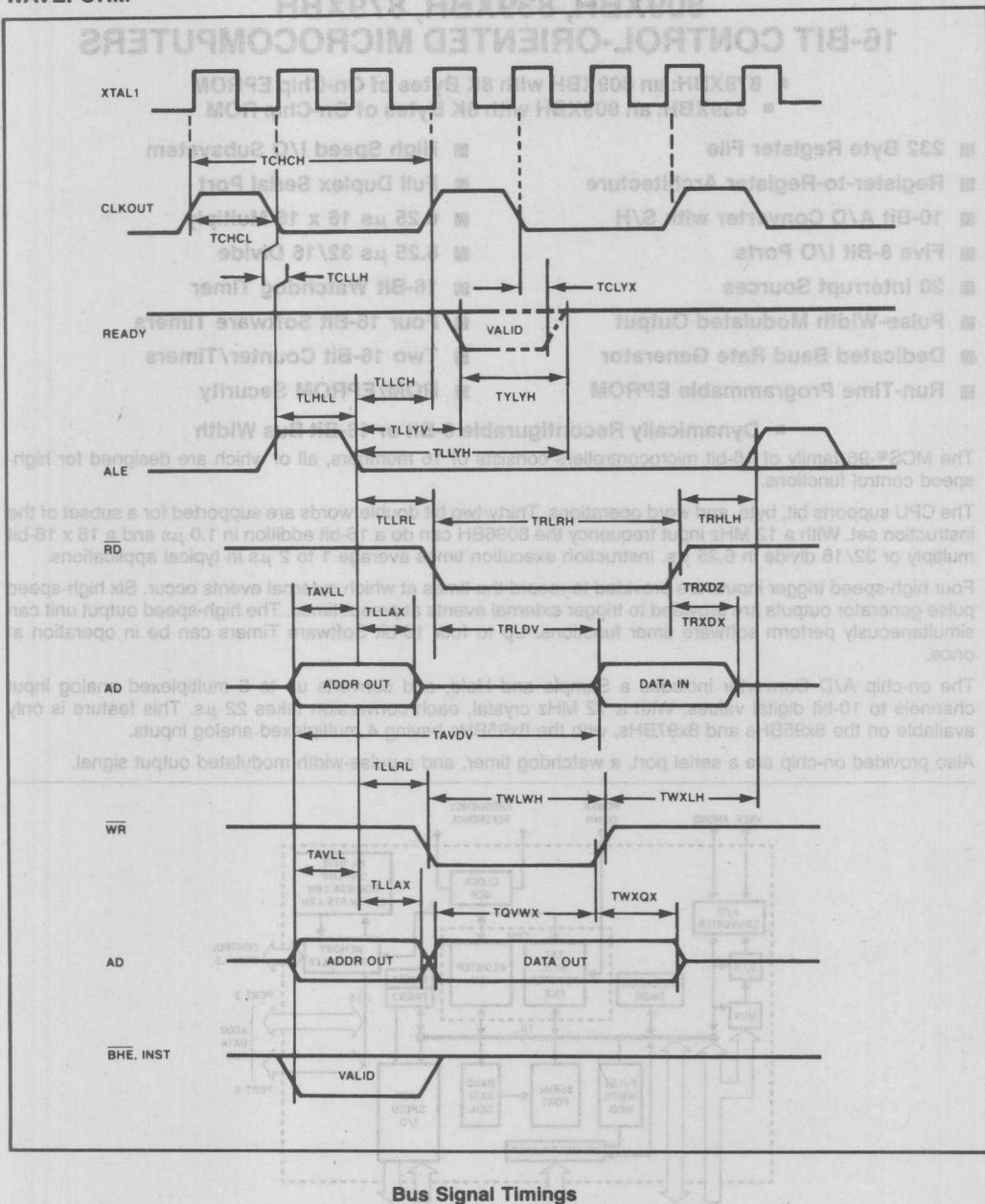
**Timing Responses** (MCS-96 parts meet these specs)

| Symbol | Parameter                         | Min                    | Max                   | Units |
|--------|-----------------------------------|------------------------|-----------------------|-------|
| FXTAL  | Oscillator Frequency              | 6.00                   | 12.00                 | MHz   |
| Tosc   | Oscillator Period                 | 83                     | 166                   | nsec  |
| TCHCH  | CLKOUT Period (3)                 | 3T <sub>osc</sub> (4)  | 3T <sub>osc</sub> (4) | nsec  |
| TCHCL  | CLKOUT High Time                  | T <sub>osc</sub> - 20  | T <sub>osc</sub> + 20 | nsec  |
| TCLLH  | CLKOUT Low to ALE High            | - 5                    | 20                    | nsec  |
| TLLCH  | ALE Low to CLKOUT High            | T <sub>osc</sub> - 20  | T <sub>osc</sub> + 40 | nsec  |
| TLHLL  | ALE Pulse Width                   | T <sub>osc</sub> - 25  | T <sub>osc</sub> + 15 | nsec  |
| TAVLL  | Address Setup to End of ALE       | T <sub>osc</sub> - 50  |                       | nsec  |
| TLLRL  | End of ALE to RD/ or WR/ active   | T <sub>osc</sub> - 20  |                       | nsec  |
| TLLAX  | Address hold after End of ALE     | T <sub>osc</sub> - 20  |                       | nsec  |
| TWLWH  | WR/ Pulse Width                   | 2T <sub>osc</sub> - 35 |                       | nsec  |
| TQVWX  | Output Data Setup to End of WR/   | 2T <sub>osc</sub> - 60 |                       | nsec  |
| TWXQX  | Output Data Hold after End of WR/ | T <sub>osc</sub> - 25  |                       | nsec  |
| TWXLH  | End of WR/ to next ALE            | 2T <sub>osc</sub> - 30 |                       | nsec  |
| TRLRH  | RD/ Pulse Width                   | 3T <sub>osc</sub> - 30 |                       | nsec  |
| TRHLH  | End of RD/ to next ALE            | T <sub>osc</sub> - 25  |                       | nsec  |

**NOTES:**

1. If the 48-pin part is being used then this timing can be generated by assuming that the CLKOUT falling edge has occurred at 2T<sub>osc</sub> + 60 (TLLCH(max) + TCHCL(max)) after the falling edge of ALE.
2. If more than one wait state is desired, add 3T<sub>osc</sub> for each additional wait state.
3. This specification is not tested, but is verified by design analysis and/or derived from other tested parameters.
4. CLKOUT is directly generated as a divide by 3 of the oscillator. The period will be 3T<sub>osc</sub> +/- 10 nsec if T<sub>osc</sub> is constant and the rise and fall times on XTAL 1 are less than 10 nsec.

WAVEFORM



Bus Signal Timings

# **MCS®-96** **809XBH, 839XBH, 879XBH** **16-BIT CONTROL-ORIENTED MICROCOMPUTERS**

- 879XBH: an 809XBH with 8K Bytes of On-Chip EPROM
- 839XBH: an 809XBH with 8K Bytes of On-Chip ROM

- 232 Byte Register File
- Register-to-Register Architecture
- 10-Bit A/D Converter with S/H
- Five 8-Bit I/O Ports
- 20 Interrupt Sources
- Pulse-Width Modulated Output
- Dedicated Baud Rate Generator
- Run-Time Programmable EPROM
- High Speed I/O Subsystem
- Full Duplex Serial Port
- 6.25  $\mu$ s 16 x 16 Multiply
- 6.25  $\mu$ s 32/16 Divide
- 16-Bit Watchdog Timer
- Four 16-Bit Software Timers
- Two 16-Bit Counter/Timers
- ROM/EPROM Security
- Dynamically Reconfigurable 8-Bit or 16-Bit Bus Width

The MCS®-96 family of 16-bit microcontrollers consists of 16 members, all of which are designed for high-speed control functions.

The CPU supports bit, byte, and word operations. Thirty-two bit double-words are supported for a subset of the instruction set. With a 12 MHz input frequency the 8096BH can do a 16-bit addition in 1.0  $\mu$ s and a 16 x 16-bit multiply or 32/16 divide in 6.25  $\mu$ s. Instruction execution times average 1 to 2  $\mu$ s in typical applications.

Four high-speed trigger inputs are provided to record the times at which external events occur. Six high-speed pulse generator outputs are provided to trigger external events at preset times. The high-speed output unit can simultaneously perform software timer functions. Up to four 16-bit Software Timers can be in operation at once.

The on-chip A/D Converter includes a Sample and Hold, and converts up to 8 multiplexed analog input channels to 10-bit digital values. With a 12 MHz crystal, each conversion takes 22  $\mu$ s. This feature is only available on the 8x95BHs and 8x97BHs, with the 8x95BHs having 4 multiplexed analog inputs.

Also provided on-chip are a serial port, a watchdog timer, and a pulse-width modulated output signal.

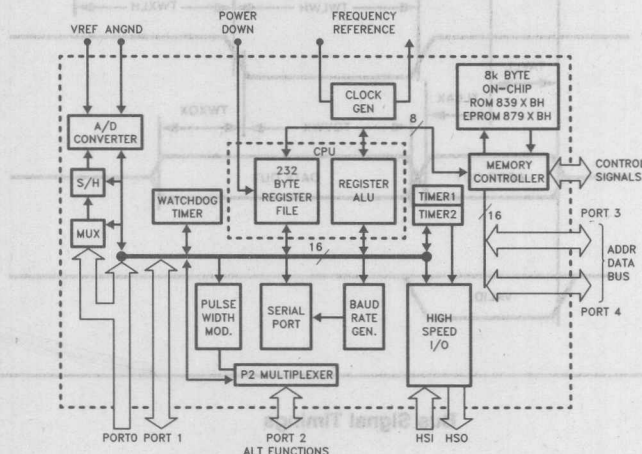


Figure 1. MCS®-96 Block Diagram

270090-1



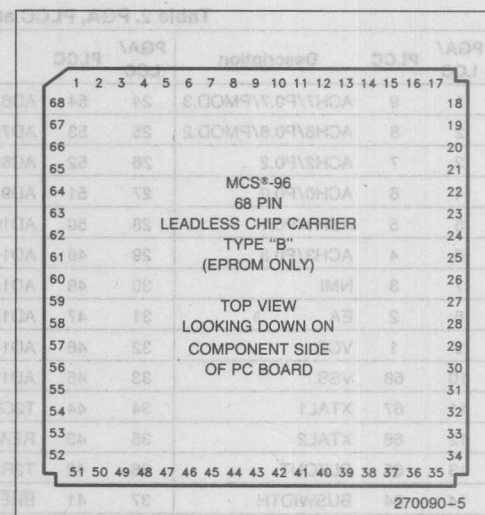
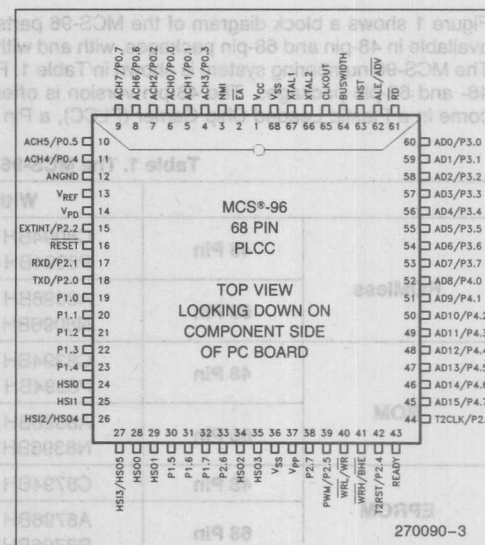
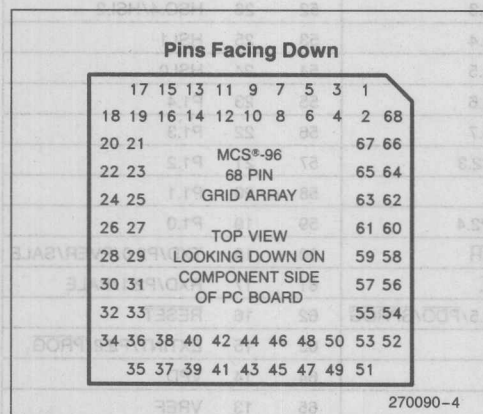
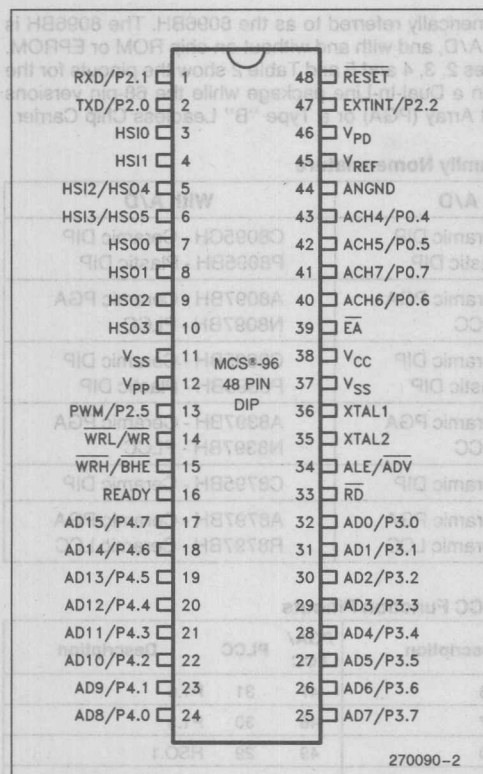
Figure 1 shows a block diagram of the MCS-96 parts, generically referred to as the 8096BH. The 8096BH is available in 48-pin and 68-pin packages, with and without A/D, and with and without on-chip ROM or EPROM. The MCS-96 numbering system is shown in Table 1. Figures 2, 3, 4 and 5 and Table 2 show the pinouts for the 48- and 68-pin packages. The 48-pin version is offered in a Dual-In-Line package while the 68-pin versions come in a Plastic Leaded Chip Carrier (PLCC), a Pin Grid Array (PGA) or a Type "B" Leadless Chip Carrier.

Table 1. The MCS-96® Family Nomenclature

|         |        | Without A/D                                    | With A/D                                       |
|---------|--------|--|--|
| ROMless | 48 Pin | C8094BH - Ceramic DIP<br>P8094BH - Plastic DIP | C8095CH - Ceramic DIP<br>P8095BH - Plastic DIP |
|         | 68 Pin | A8096BH - Ceramic PGA<br>N8096BH - PLCC        | A8097BH - Ceramic PGA<br>N8097BH - PLCC        |
| ROM     | 48 Pin | C8394BH - Ceramic DIP<br>P8394BH - Plastic DIP | C8395BH - Ceramic DIP<br>P8395BH - Plastic DIP |
|         | 68 Pin | A8396BH - Ceramic PGA<br>N8396BH - PLCC        | A8397BH - Ceramic PGA<br>N8397BH - PLCC        |
| EPROM   | 48 Pin | C8794BH - Ceramic DIP                          | C8795BH - Ceramic DIP                          |
|         | 68 Pin | A8796BH - Ceramic PGA<br>R8796BH - Ceramic LCC | A8797BH - Ceramic PGA<br>R8797BH - Ceramic LCC |

Table 2. PGA, PLCC and LCC Function Pinouts

| PGA/<br>LCC | PLCC | Description      | PGA/<br>LCC | PLCC | Description        | PGA/<br>LCC | PLCC | Description        |
|-------------|------|------------------|-------------|------|--------------------|-------------|------|--------------------|
| 1           | 9    | ACH7/P0.7/PMOD.3 | 24          | 54   | AD6/P3.6           | 47          | 31   | P1.6               |
| 2           | 8    | ACH6/P0.6/PMOD.2 | 25          | 53   | AD7/P3.7           | 48          | 30   | P1.5               |
| 3           | 7    | ACH2/P0.2        | 26          | 52   | AD8/P4.0           | 49          | 29   | HSO.1              |
| 4           | 6    | ACH0/P0.0        | 27          | 51   | AD9/P4.1           | 50          | 28   | HSO.0              |
| 5           | 5    | ACH1/P0.1        | 28          | 50   | AD10/P4.2          | 51          | 27   | HSO.5/HSI.3        |
| 6           | 4    | ACH3/P0.3        | 29          | 49   | AD11/P4.3          | 52          | 26   | HSO.4/HSI.2        |
| 7           | 3    | NMI              | 30          | 48   | AD12/P4.4          | 53          | 25   | HSI.1              |
| 8           | 2    | EA               | 31          | 47   | AD13/P4.5          | 54          | 24   | HSI.0              |
| 9           | 1    | VCC              | 32          | 46   | AD14/P4.6          | 55          | 23   | P1.4               |
| 10          | 68   | VSS              | 33          | 45   | AD15/P4.7          | 56          | 22   | P1.3               |
| 11          | 67   | XTAL1            | 34          | 44   | T2CLK/P2.3         | 57          | 21   | P1.2               |
| 12          | 66   | XTAL2            | 35          | 43   | READY              | 58          | 20   | P1.1               |
| 13          | 65   | CLKOUT           | 36          | 42   | T2RST/P2.4         | 59          | 19   | P1.0               |
| 14          | 64   | BUSWIDTH         | 37          | 41   | BHE/WRH            | 60          | 18   | TXD/P2.0/PVER/SALE |
| 15          | 63   | INST             | 38          | 40   | WR/WRL             | 61          | 17   | RXD/P2.1/PALE      |
| 16          | 62   | ALE/ADV          | 39          | 39   | PWM/P2.5/PDO/SPROG | 62          | 16   | RESET              |
| 17          | 61   | RD               | 40          | 38   | P2.7               | 63          | 15   | EXTINT/P2.2/PROG   |
| 18          | 60   | AD0/P3.0         | 41          | 37   | VPP                | 64          | 14   | VPD                |
| 19          | 59   | AD1/P3.1         | 42          | 36   | VSS                | 65          | 13   | VREF               |
| 20          | 58   | AD2/P3.2         | 43          | 35   | HSO.3              | 66          | 12   | ANGND              |
| 21          | 57   | AD3/P3.3         | 44          | 34   | HSO.2              | 67          | 11   | ACH4/P0.4/PMOD.0   |
| 22          | 56   | AD4/P3.4         | 45          | 33   | P2.6               | 68          | 10   | ACH5/P0.5/PMOD.1   |
| 23          | 55   | AD5/P3.5         | 46          | 32   | P1.7               |             |      |                    |



## FUNCTIONAL OVERVIEW

The following section is an overview of the 8096BH. Additional information is available in the Microcontroller Handbook, order number 230843-002.

## CPU Architecture

The 8096BH uses the same address space for both program and data memory, except in the address range from 00H through 0FFH. Data fetches in this range are always to the Register File, while instruction fetches from these locations are directed to external memory. (Locations 00H through 0FFH in external memory are reserved for Intel development systems).

Within the Register File, locations 00H through 17H are register mapped I/O control registers, also referred to as Special Function Registers (SFRs). The rest of the Register File (018H through 0FFH) contains 232 bytes of RAM, which can be referenced as bytes, words, or double-words. This register space allows the user to keep the most frequently-used variables in on-chip RAM, which can be accessed faster than external memory. Locations 0F0H through 0FFH can be preserved during power down via a separate power down pin ( $V_{PD}$ ).

Outside of the register file, program memory, data memory, and peripherals can be intermixed. The addresses with special significance are:

|             |                               |
|-------------|-------------------------------|
| 0000H-0017H | Register Mapped I/O (SFRs)    |
| 0018H-0019H | Stack Pointer                 |
| 1FFEh-1FFFh | Ports 3 and 4                 |
| 2000H-2011H | Interrupt Vectors             |
| 2012H-2017H | Reserved                      |
| 2018H       | Chip Configuration Byte       |
| 2019H       | Reserved                      |
| 201AH-201BH | "Jump to Self" Opcode (27 FE) |
| 201CH-201FH | Reserved                      |
| 2020H-202FH | Security Key                  |
| 2030H-207FH | Reserved                      |
| 2080H       | Reset Location                |

The 839XBH carries 8K bytes of ROM, while the 879XBH has 8K bytes of EPROM. With ROM and EPROM parts, the internal program memory occu-

pies addresses 2000H through 3FFFFH. Instruction or data fetches from these addresses access the on-chip memory if the EA pin is externally held at a logical 1. If the EA pin is at a logical 0, these addresses access off-chip memory. On the 879XBH parts, holding EA at +12.5V puts the part in Programming Mode, which is described in the EPROM Characteristics Section of this data sheet.

A memory map for the MCS-96 product family is shown in Figure 6.

The RALU (Register/ALU) section consists of a 17-bit ALU, the Program Status Word, the Program Counter, and several temporary registers. A key feature of the 8096BH is that it does not use an accumulator. Rather, it operates directly on any register in the Register File. Being able to operate directly on data in the Register File without having to move it into and out of an accumulator results in a significant improvement in execution speed.

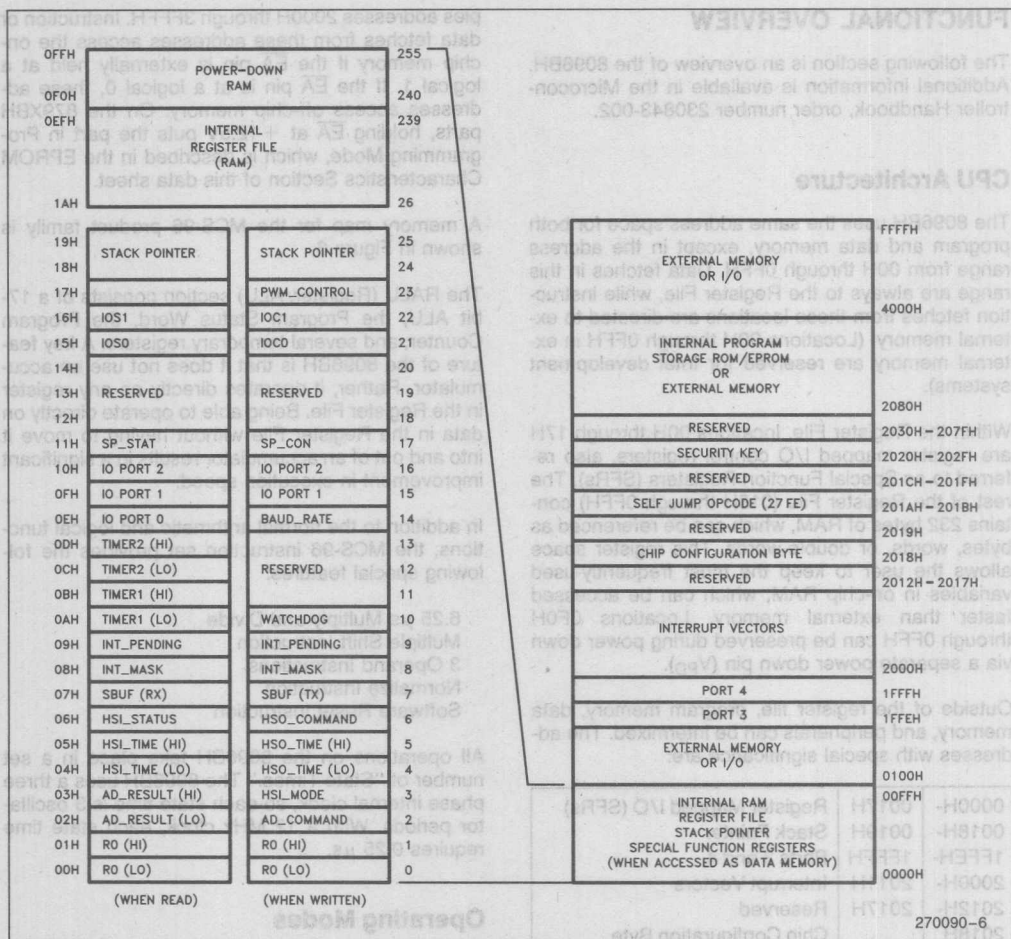
In addition to the normal arithmetic and logical functions, the MCS-96 instruction set provides the following special features:

- 6.25  $\mu$ s Multiply and Divide
- Multiple Shift Instruction
- 3 Operand Instructions
- Normalize Instruction
- Software Reset Instruction

All operations on the 8096BH take place in a set number of "State Times." The 8096BH uses a three phase internal clock, so each state time is 3 oscillator periods. With a 12 MHz clock, each state time requires 0.25  $\mu$ s.

## Operating Modes

The 8096BH supports a variety of options to simplify memory systems, interfacing requirements and ready control. Bus flexibility is provided by allowing selection of bus control signal definitions and run-time selection of the external bus width. In addition, several Ready control modes are available to simplify the external hardware requirements for accessing slow devices. The Chip Configuration Register is used to store the operating mode information.



### Figure 6. Memory Map



## CHIP CONFIGURATION REGISTER (CCR)

Configuration information is stored in the Chip Configuration Register (CCR). Four of the bits in the register specify the bus control mode and ready control mode. Two bits also govern the level of ROM/EPROM protection and one bit is Nanded with the BUSWIDTH pin every bus cycle to determine the bus size. The CCR bit map is shown in Figure 7, and the functions associated with each bit are described later.

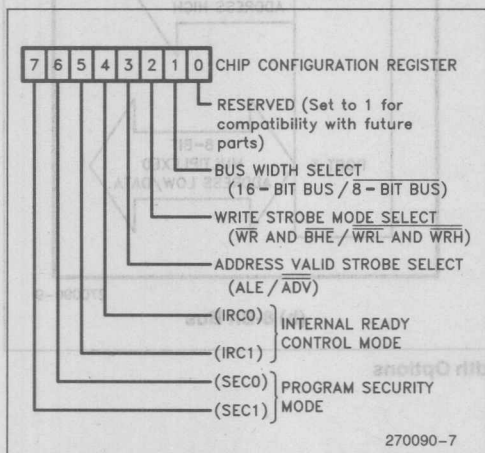


Figure 7. Chip Configuration Register

The CCR is loaded on reset with the Chip Configuration Byte, located at address 2018H. The CCR register is a non-memory mapped location that can only be written to during the reset sequence; once it is loaded it cannot be changed until the next reset occurs. The 8096BH will correctly read this location in every bus mode.

If the  $\overline{EA}$  pin is set to a logical 0, the access to 2018H comes from external memory. If  $\overline{EA}$  is a logical 1, the access comes from internal ROM/EPROM. If  $\overline{EA}$  is +12.5V, the CCR is loaded with a byte from a separate non-memory-mapped location called PCCB (Programming CCB). The Programming mode is described in the EPROM Characteristics Section.

## BUS WIDTH

The 8096BH external-bus width can be run-time configured to operate as a standard 16-bit multi-

plexed address/data bus, or as an 8088 minimum mode type 16-bit address/ 8-bit data bus.

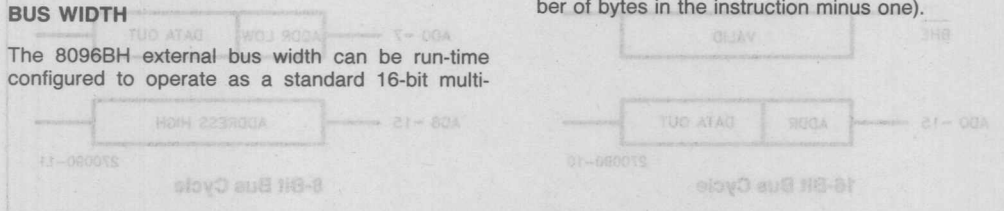
During 16-bit bus cycles, Ports 3 and 4 contain the address multiplexed with data using ALE to latch the address. In 8-bit bus cycles, Port 3 is multiplexed address/data while Port 4 is address bits 8 through 15. The address bits on Port 4 are valid throughout an 8-bit bus cycle. Figure 8 shows the two options.

The bus width can be changed each bus cycle and is controlled using bit 1 of the CCR with the BUSWIDTH pin. If either CCR.1 or BUSWIDTH is a 0, external accesses will be over a 16-bit address/8-bit data bus. If both CCR.1 and BUSWIDTH are 1s, external accesses will be over a 16-bit address/16-bit data bus. Internal accesses are always 16-bits wide.

The bus width can be changed every external bus cycle if a 1 was loaded into CCR bit 1 at reset. If this is the case, changing the value of the BUSWIDTH pin at run-time will dynamically select the bus width. For example, the user could feed the INST line into the BUSWIDTH pin, thus causing instruction accesses to be word wide from EPROMs while data accesses are byte wide to and from RAMs. A second example would be to place an inverted version of Address bit 15 on the BUSWIDTH pin. This would make half of external memory word wide, while half is byte wide.

Since BUSWIDTH is sampled after address decoding has had time to occur, even more complex memory maps could be constructed. See the timing specifications for an exact description of BUSWIDTH timings. The bus width will be determined by bit 1 of the CCR alone on 48-pin parts since they do not have a BUSWIDTH pin.

When using an 8-bit bus, some performance degradation is to be expected. On the 8096BH, instruction execution times with an 8-bit bus will slow down if any of three conditions occur. First, word writes to external memory will cause the executing instruction to take two extra state times to complete. Second, word reads from external memory will cause a one state time extension of instruction execution time. Finally, if the prefetch queue is empty when an instruction fetch is requested, instruction execution is lengthened by one state time for each byte that must be externally acquired (worst case is the number of bytes in the instruction minus one).



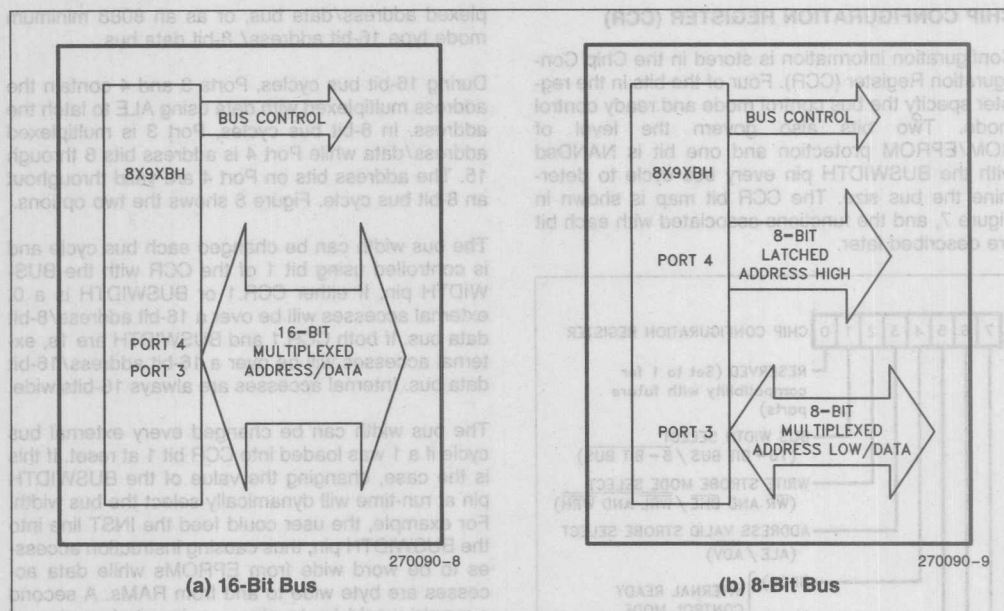


Figure 8. Bus Width Options

## BUS CONTROL

The 8096BH can be made to provide bus control signals of several types. Three control lines have dual functions designed to reduce external hardware. Bits 2 and 3 of the CCR specify the functions performed by these control lines.

### Standard Bus Control

If CCR bits 2 and 3 are 1s, then the standard 8096BH control signals  $\overline{WR}$ ,  $\overline{BHE}$  and ALE are provided (Figure 9).  $\overline{WR}$  will come out for every write.  $\overline{BHE}$  will be valid throughout the bus cycle and can be combined with  $\overline{WR}$  and address line 0 to form  $\overline{WRL}$  and  $\overline{WRH}$ . ALE will rise as the address starts to come out, and will fall to provide the signal to externally latch the address.

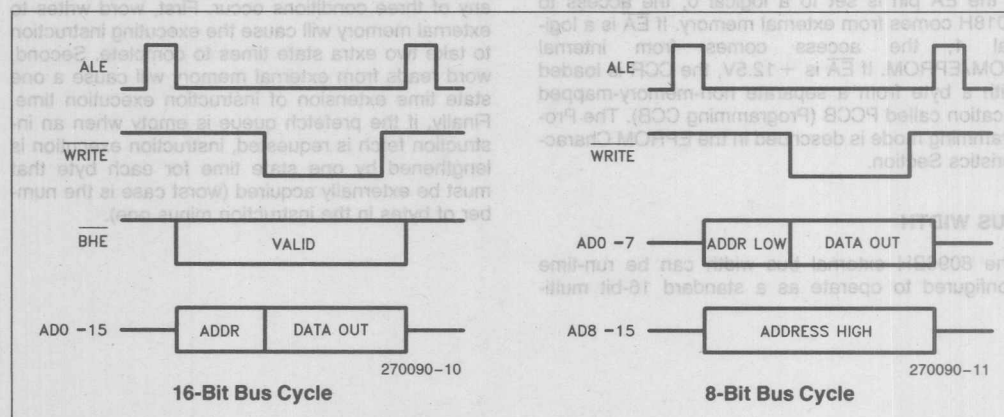


Figure 9. Standard Bus Control

# Write Strobe Mode

The Write Strobe Mode eliminates the necessity to externally decode for odd or even byte writes. If CCR bit 2 is a 0, and the bus is in a 16-bit cycle,  $\overline{WRL}$  and  $\overline{WRH}$  signals are provided in place of  $\overline{WR}$  and  $\overline{BHE}$  (Figure 10).  $\overline{WRL}$  will go low for all byte writes to an even address and all word writes.  $\overline{WRH}$  will go low for all byte writes to an odd address and all word writes.

In an 8-bit bus cycle with CCR bit 2 a 0,  $\overline{WR}$  is provided (Figure 10).  $\overline{WR}$  will go low for all writes.

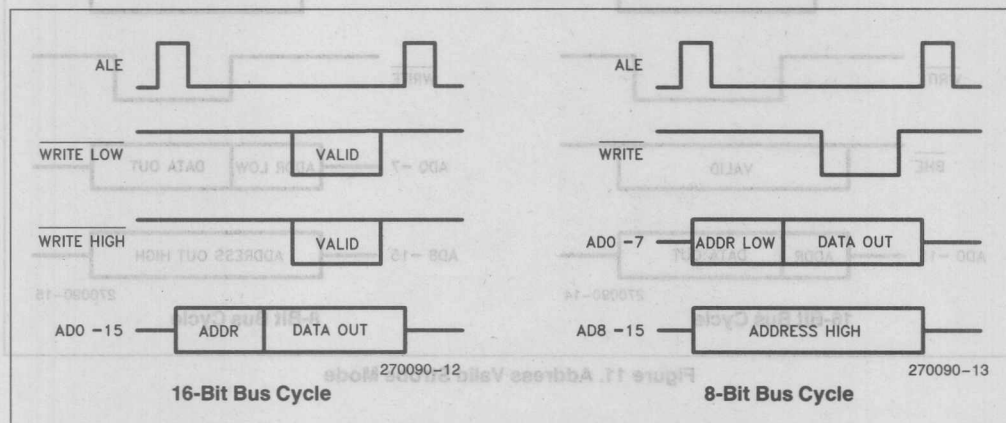


Figure 10. Write Strobe Mode

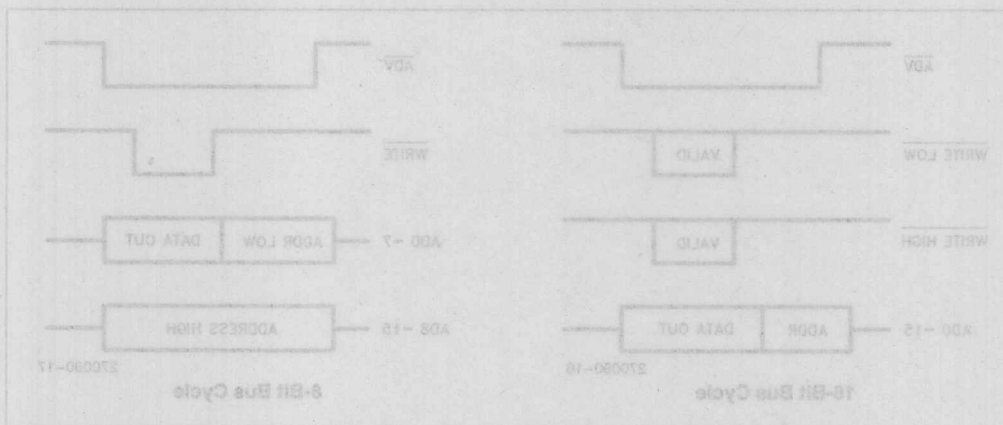


Figure 12. Write Strobe Mode with Address Valid Strobe

### Address Valid Strobe Mode

If CCR bit 3 is a 0, then an Address Valid strobe is provided in the place of ALE (Figure 11). When the address valid mode is selected,  $\overline{ADV}$  will go low after an external address is set up. It will stay low until the end of the bus cycle, where it will go inactive high. This can be used to provide a chip select for external memory.

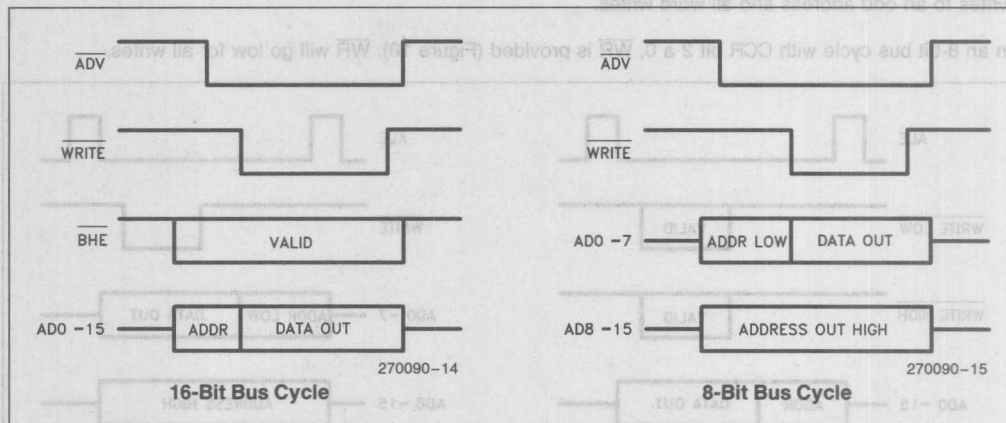


Figure 11. Address Valid Strobe Mode

### Address Valid with Write Strobe

If both CCR bits 2 and 3 are 0s, both the Address Valid strobe and the Write Strobes will be provided for bus control. Figure 12 shows these signals.

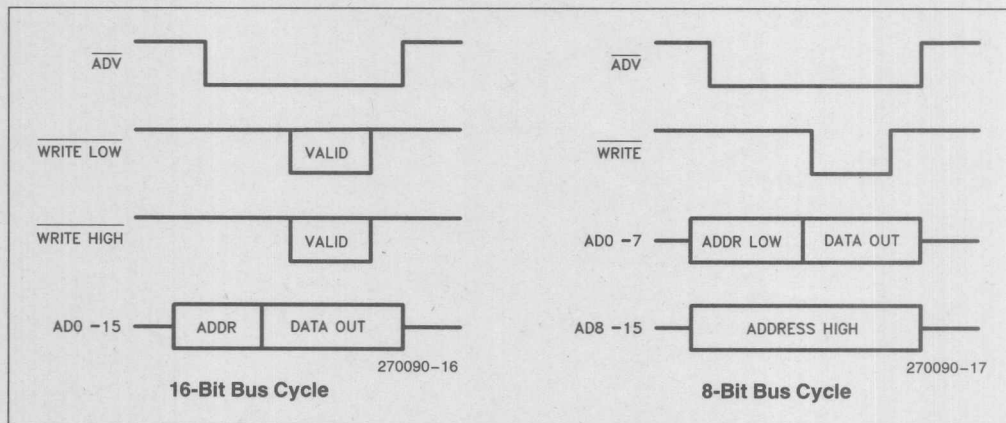


Figure 12. Write Strobe with Address Valid Strobe



## READY CONTROL

To simplify ready control, four modes of internal ready control logic have been provided. The modes are chosen by properly configuring bits 4 and 5 of the CCR.

The internal ready control logic can be used to limit the number of wait states that slow devices can insert into the bus cycle. When the READY pin is pulled low, wait states will be inserted into the bus cycle until the READY pin goes high, or the number of wait states equals the number specified by CCR bits 4 and 5, whichever comes first. Table 3 shows the number of wait states that can be selected. Internal Ready control can be disabled by loading 11 into bits 4 and 5 of the CCR.

Table 3. Internal Ready Control

| IRC1 | IRC0 | Description                    |
|------|------|--------------------------------|
| 0    | 0    | Limit to 1 Wait State          |
| 0    | 1    | Limit to 2 Wait States         |
| 1    | 0    | Limit to 3 Wait States         |
| 1    | 1    | Disable Internal Ready Control |

This feature provides for simple ready control. For example, every slow memory chip select line could be ORed together and be connected to the READY pin with CCR bits 4 and 5 programmed to give the proper number of wait states to the slow devices.

## PROGRAM SECURITY

Four modes of program security are available on the 839XBH and 879XBH parts. CCR bits 6 and 7 (SEC0, SEC1) select whether internal program memory can be read (or written in EPROM parts) by a program executing from external memory. The modes are shown in Table 4. Internal ROM/EPROM addresses 2020H through 3FFFH are protected from reads while 2000H through 3FFFH are protected from writes, as set by the CCR.

Table 4. Program Security Modes

| SEC1 | SEC0 | Protection               |
|------|------|--------------------------|
| 0    | 0    | Read and Write Protected |
| 0    | 1    | Write Protected          |
| 1    | 0    | Read Protected           |
| 1    | 1    | No Protection            |

Only code executing from internal memory can read protected internal memory, while a write protected memory can not be written to, even from internal execution. As a result of 8096BH prefetching of instructions, however, accesses to protected memory are not allowed for instructions located above 3FFAH. Note that the interrupt vectors and the CCR are not protected.

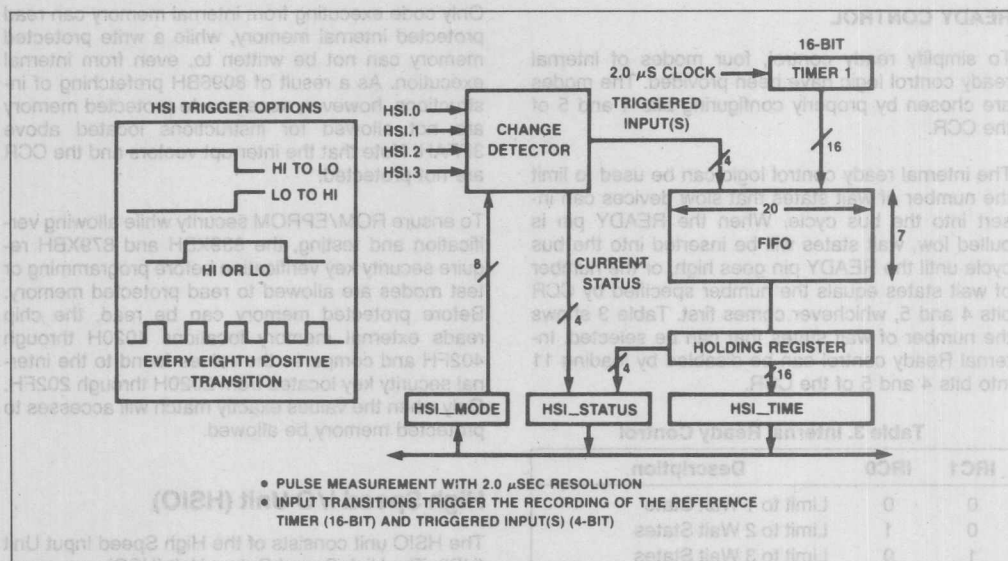
To ensure ROM/EPROM security while allowing verification and testing, the 839XBH and 879XBH require security key verification before programming or test modes are allowed to read protected memory. Before protected memory can be read, the chip reads external memory locations 4020H through 402FH and compares the values found to the internal security key located from 2020H through 202FH. Only when the values exactly match will accesses to protected memory be allowed.

## High Speed I/O Unit (HSIO)

The HSIO unit consists of the High Speed Input Unit (HSI), The High Speed Output Unit (HSO), one counter and one timer. "High Speed" denotes that the units can perform functions related to the timers without CPU intervention. The HSI records times when events occur and the HSO triggers events at pre-programmed times.

All actions within the HSIO unit are synchronized to the timers. The two 16-bit timer/counter registers in the HSIO unit are cleared on chip reset and can be programmed to generate an interrupt on overflow. The Timer 1 register is automatically incremented every 8 state times (every 2.0  $\mu$ s, with a 12 MHz clock). The Timer 2 register can be programmed to count transitions on either the T2CLK pin or HSI.1 pin. It is incremented on both positive and negative edges of the selected input line. In addition to being cleared by reset, Timer 2 can also be cleared in software or by signals from input pins T2RST or HSI.0. Neither of these timers is required for either the Watchdog timer or the serial port.

The High Speed Input (HSI) unit can detect transitions on any of its 4 input lines. When one occurs it records the time (from Timer 1) and which input lines made the transition. This information is recorded with 2  $\mu$ s (12 MHz system) resolution and stored in an 8-level FIFO. The unit can be programmed to look for four types of events, as shown in Figure 13. It can activate the HSI Data Available interrupt either when the Holding Register is loaded or the 6th FIFO entry has been made. Each input line can be individually enabled or disabled to the HSI unit by software.



**Figure 13. High Speed Input Unit**

stored in the CAM (Content Addressable Memory) file at any one time. As each action is carried out at its preset time that command is removed from the CAM making space for another command. HSO.4 and HSO.5 are shared with the HSI unit as HSI.2 and HSI.3, and can be individually enabled or disabled as outputs.

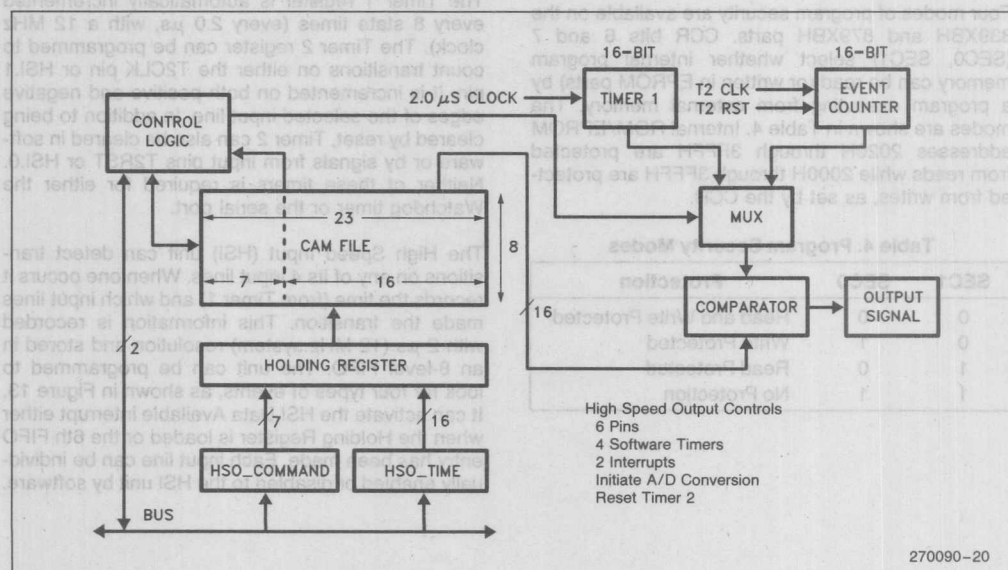


Figure 14. High Speed Output Unit

## Standard I/O Ports

There are 5 8-bit I/O ports on the 8096BH in addition to the High Speed I/O lines.

Port 0 is an input-only port which shares its pins with the analog inputs to the A/D Converter. The port can be read digitally and/or, by writing to the A/D Command Register, one of the lines can be selected as the input to the A/D Converter. Port 0 is also used to input mode information on EPROM parts operating in the Programming mode.

Port 1 is a quasi-bidirectional I/O port. "Quasi-bidirectional" means the port pin has a weak internal pullup that is always active and an internal pulldown which can either be on (to output a 0) or off (to output a 1). This configuration allows the pin to be used as either an input or an output without using a data direction register. In parallel with the weak internal pullup is a much stronger internal pulldown that is activated for one state time when the pin is internally driven from 0 to 1. This is done to speed up the 0-to-1 transition time.

Port 2 is a multi-functional port. Two of the pins (P2.6, 2.7) are quasi-bidirectional while the remaining six are shared with other functions in the 8096BH, as shown in Table 5. Port 2 is also used for control signals by EPROM parts operating in the Programming Mode.

Table 5. Port 2 Pin Functions

| Port | Function | Alternate Function           |
|------|----------|------------------------------|
| P2.0 | output   | TXD (serial port transmit)   |
| P2.1 | input    | RXD (serial port receive)    |
| P2.2 | input    | EXTINT (external interrupt)  |
| P2.3 | input    | T2CLK (Timer 2 clock)        |
| P2.4 | input    | T2RST (Timer 2 reset)        |
| P2.5 | output   | PWM (pulse-width modulation) |

Ports 3 and 4 are bi-directional I/O ports with open drain outputs. These pins are also used as the multiplexed address/data bus when accessing external memory, in which case they have strong internal pullups. The internal pullups are only used during external memory read or write cycles when the pins are outputting address or data bits. At any other time, the internal pullups are disabled. When used as a system bus, Ports 3 and 4 can be configured to be either a multiplexed 16-bit address/data bus or a multiplexed 16-bit address/ 8-bit data bus. EPROM parts also use Ports 3 and 4 to pass programming commands, addresses, data and status.

## Serial Port

The serial port is compatible with the MCS-51 family, (8051, 8031 etc.), serial port. It is full duplex, and double-buffered on receive. There are 3 asynchronous modes and 1 synchronous mode of operation for the serial port. The asynchronous modes allow for 8 or 9 bits of data with even parity optionally inserted for one of the data bits. Selective interrupts based on the 9th data bit are available to support interprocessor communication.

Baud rates in all modes are determined by an independent 16-bit on-chip baud rate generator. Either the XTAL1 pin or the T2CLK pin can be used as the input to the baud rate generator. The maximum baud rate in the asynchronous mode is 187.5 KBaud. The maximum baud rate in the synchronous mode is 1.5 MBaud.

## Pulse Width Modulator (PWM)

The PWM output shares a pin with port bit P2.5. When the PWM output is selected, this pin outputs a pulse train having a fixed period of 256 state times, and a programmable width of 0 to 255 state times. The width is programmed by loading the desired value, in state times, to the PWM Control Register.

## A/D Converter with Sample and Hold

The analog-to-digital converter is a 10-bit, successive approximation converter with internal sample and hold. It has a fixed conversion time of 88 state times which includes the 4 state acquisition time of the internal Sample/Hold. With a 12 MHz clock, the conversion takes 22  $\mu$ s, including the 1  $\mu$ s sample for the Sample and Hold. The Sample acquisition begins 4 state times after the conversion is triggered. A 2 pF capacitance is charged from the input signal during acquisition.

The analog input must be in the range of 0 to  $V_{REF}$  (nominally,  $V_{REF} = 5V$ ). This input can be selected from 8 analog input lines, which connect to the same pins as Port 0. A conversion can be initiated either by setting a control bit in the A/D Command register, or by programming the HSO unit to trigger the conversion at some specified time.

## Interrupts

The 8096BH has 20 interrupt sources which vector through 8 interrupt vectors. A 0-to-1 transition from

any of the sources sets a corresponding bit in the Interrupt Pending register. The content of the Interrupt Mask register determines if a pending interrupt will be serviced or not. If it is to be serviced, the CPU pushes the current Program Counter onto the Stack and reloads it with the vector corresponding to the desired interrupt. The interrupt vectors are located in addresses 2000H through 2011H, as shown in Figure 15.

| Vector         | Vector Location |            | Priority       |
|----------------|-----------------|------------|----------------|
|                | (High Byte)     | (Low Byte) |                |
| Software       | 2011H           | 2010H      | Not Applicable |
| Extint         | 200FH           | 200EH      | 7 (Highest)    |
| Serial Port    | 200DH           | 200CH      | 6              |
| Software       | 200BH           | 200AH      | 5              |
| Timers         |                 |            |                |
| HSI.0          | 2009H           | 2008H      | 4              |
| High Speed     | 2007H           | 2006H      | 3              |
| Outputs        |                 |            |                |
| HSI Data       | 2005H           | 2004H      | 2              |
| Available      |                 |            |                |
| A/D Conversion | 2003H           | 2002H      | 1              |
| Complete       |                 |            |                |
| Timer Overflow | 2001H           | 2000H      | 0 (Lowest)     |

Figure 15. Interrupt Vectors

At the end of the interrupt routine the RET instruction pops the program counter from the stack and execution continues where it left off. It is not necessary to store and replace registers during interrupt routines as each routine can be set up to use a dif-

ferent section of the register file. This feature of the architecture provides for very fast context switching. While the 8096BH has a single priority level in the sense that any interrupt may itself be interrupted, a priority structure exists for resolving simultaneously pending interrupts, as indicated in Figure 15. Since the interrupt pending and interrupt mask registers can be manipulated in software, it is possible to dynamically alter the interrupt priorities to suit the users software.

## Watchdog Timer

The watchdog timer is a 16-bit counter which, once started, is incremented every state time. If not cleared before it overflows, the RESET pin will be pulled down for two state times, causing the system to be reinitialized. In a 12 MHz system, the watchdog timer overflows after 16 ms.

This feature is provided as a means of graceful recovery from a software upset. The counter must be cleared by the software before it overflows, or else the system assumes an upset has occurred and activates RESET. Once the watchdog timer is started it cannot be turned off by software. The flip-flop which enables the watchdog timer has been designed to maintain its state through  $V_{CC}$  glitches to as low as 0V or as high as 7V for 1  $\mu$ s to 1 ms.

To start the watchdog timer, or to clear it, one writes 1EH followed by 0E1H to the WDT address (000AH). The Watchdog cannot be stopped once it is started unless the system is reset.

|      |        |                              |
|------|--------|------------------------------|
| P2.0 | output | TXD (serial port transmit)   |
| P2.1 | input  | RXD (serial port receive)    |
| P2.2 | input  | EXTINT (external interrupt)  |
| P2.3 | input  | T2CLK (Timer 2 clock)        |
| P2.4 | input  | T2RST (Timer 2 reset)        |
| P2.5 | output | PWM (pulse-width modulation) |

Ports 3 and 4 are bi-directional I/O ports with open drain outputs. These pins are also used as the multiplexed address/data bus when accessing external memory, in which case they have strong internal pullups. The internal pullups are only used during memory read or write cycles when the pins are outputting address or data bits. At any other time, the internal pullups are disabled. When used as a system bus, Ports 3 and 4 can be configured to be either a multiplexed 16-bit address/data bus or a multiplexed 16-bit address/8-bit data bus. EPROM pins also use Ports 3 and 4 to pass programming commands, addresses, data and status.



## PIN DESCRIPTIONS

| Symbol           | Name and Function   |
|------------------|---|
| V <sub>CC</sub>  | Main supply voltage (5V).   |
| V <sub>SS</sub>  | Digital circuit ground (0V).  |
| V <sub>PD</sub>  | RAM standby supply voltage (5V). This voltage must be present during normal operation. In a Power Down condition (i.e. V <sub>CC</sub> drops to zero), if RESET is activated before V <sub>CC</sub> drops below spec and V <sub>PD</sub> continues to be held within spec., the top 16 bytes in the Register File will retain their contents. RESET must be held low during the Power Down and should not be brought high until V <sub>CC</sub> is within spec and the oscillator has stabilized. |
| V <sub>REF</sub> | Reference voltage for the A/D converter (5V). V <sub>REF</sub> is also the supply voltage to the analog portion of the A/D converter and the logic used to read Port 0.   |
| ANGND            | Reference ground for the A/D converter. Should be held at nominally the same potential as V <sub>SS</sub> .   |
| V <sub>PP</sub>  | Programming voltage for the EPROM parts. It should be +12.5V. This pin is V <sub>BB</sub> on 8X9X-90 parts. Systems that have this pin connected to ANGND through a capacitance (required on 8X9X-90 parts) do not need to change.  |
| XTAL1            | Input of the oscillator inverter and of the internal clock generator.   |
| XTAL2            | Output of the oscillator inverter.  |
| CLKOUT           | Output of the internal clock generator. The frequency of CLKOUT is 1/3 the oscillator frequency. It has a 33% duty cycle.   |
| RESET            | Reset input to the chip. Input low for at least 2 state times to reset the chip. The subsequent low-to-high transition re-synchronizes CLKOUT and commences a 10-state-time sequence in which the PSW is cleared, a byte read from 2018H loads CCR, and a jump to location 2080H is executed. Input high for normal operation. RESET has an internal pullup.  |
| BUSWIDTH         | Input for buswidth selection. If CCR bit 1 is a one, this pin selects the bus width for the bus cycle in progress. If BUSWIDTH is a 1, a 16-bit bus cycle occurs. If BUSWIDTH is a 0 an 8-bit cycle occurs. If CCR bit 1 is a 0, the bus is always an 8-bit bus. This pin is the TEST pin on 8X9X-90 parts. Systems with TEST tied to V <sub>CC</sub> do not need to change. If this pin is left unconnected, it will rise to V <sub>CC</sub> .   |
| NMI              | A positive transition clears the watchdog timer count, and causes a vector to external memory location 0000H. External memory from 00H through 0FFH is reserved for Intel development systems.  |
| INST             | Output high during an external memory read indicates the read is an instruction fetch. INST is valid throughout the bus cycle.  |
| EA               | Input for memory select (External Access). EA equal to a TTL-high causes memory accesses to locations 2000H through 3FFFFH to be directed to on-chip ROM/EPROM. EA equal to a TTL-low causes accesses to these locations to be directed to off-chip memory. EA = +12.5V causes execution to begin in the Programming mode. EA has an internal pulldown, so it goes to 0 unless driven otherwise.  |
| ALE/ADV          | Address Latch Enable or Address Valid output, as selected by CCR. Both pin options provide a latch to demultiplex the address from the address/data bus. When the pin is ADV, it goes inactive high at the end of the bus cycle. ADV can be used as a chip select for external memory. ALE/ADV is activated only during external memory accesses.   |
| RD               | Read signal output to external memory. RD is activated only during external memory reads.   |
| WR/WRL           | Write and Write Low output to external memory, as selected by the CCR. WR will go low for every external write, while WRL will go low only for external writes where an even byte is being written. WR/WRL is activated only during external memory writes.   |

## PIN DESCRIPTIONS (Continued)

| Symbol        | Name and Function  |
|---------------|--|
| BHE/WRH       | Bus High Enable or Write High output to external memory, as selected by the CCR. $\overline{\text{BHE}} = 0$ selects the bank of memory that is connected to the high byte of the data bus. $\text{A0} = 0$ selects the bank of memory that is connected to the low byte of the data bus. Thus accesses to a 16-bit wide memory can be to the low byte only ( $\text{A0} = 0$ , $\overline{\text{BHE}} = 1$ ), to the high byte only ( $\text{A0} = 1$ , $\overline{\text{BHE}} = 0$ ), or both bytes ( $\text{A0} = 0$ , $\overline{\text{BHE}} = 0$ ). If the WRH function is selected, the pin will go low if the bus cycle is writing to an odd memory location.                                 |
| READY         | Ready input to lengthen external memory cycles, for interfacing to slow or dynamic memory, or for bus sharing. If the pin is high, CPU operation continues in a normal manner. If the pin is low prior to the falling edge of CLKOUT, the Memory Controller goes into a wait mode until the next positive transition in CLKOUT occurs with READY high. The bus cycle can be lengthened by up to 1 $\mu\text{s}$ . When the external memory is not being used, READY has no effect. Internal control of the number of wait states inserted into a bus cycle held not ready is available through configuration of CCR. READY has a weak internal pullup, so it goes to 1 unless externally pulled low. |
| HSI           | Inputs to High Speed Input Unit. Four HSI pins are available: HSI.0, HSI.1, HSI.2, and HSI.3. Two of them (HSI.2 and HSI.3) are shared with the HSO Unit. The HSI pins are also used as inputs by EPROM parts in Programming mode.   |
| HSO           | Outputs from High Speed Output Unit. Six HSO pins are available: HSO.0, HSO.1, HSO.2, HSO.3, HSO.4, and HSO.5. Two of them (HSO.4 and HSO.5) are shared with the HSI Unit.   |
| Port 0        | 8-bit high impedance input-only port. These pins can be used as digital inputs and/or as analog inputs to the on-chip A/D converter. These pins are also a mode input to EPROM parts in the programming mode.  |
| Port 1        | 8-bit quasi-bidirectional I/O port.  |
| Port 2        | 8-bit multi-functional port. Six of its pins are shared with other functions in the 8096BH, the remaining 2 are quasi-bidirectional. These pins are also used to input and output control signals on EPROM parts in Programming Mode.  |
| Ports 3 and 4 | 8-bit bi-directional I/O ports with open drain outputs. These pins are shared with the multiplexed address/data bus which has strong internal pullups. Ports 3 and 4 are also used as a command, address and data path by EPROM parts operating in the programming mode.   |

## INSTRUCTION SET

The 8096BH instruction set makes use of six addressing modes as described below:

**DIRECT**—The operand is specified by an 8-bit address field in the instruction. The operand must be in the Register File or SFR space (locations 0000H through 00FFH).

**IMMEDIATE**—The operand itself follows the opcode in the instruction stream as immediate data. The immediate data can be either 8-bits or 16-bits as required by the opcode.

**INDIRECT**—An 8-bit address field in the instruction gives the word address of a word register in the Register File which contains the 16-bit address of the operand. The operand can be anywhere in memory.

**INDIRECT WITH AUTO-INCREMENT**—Same as Indirect, except that, after the operand is referenced, the word register that contains the operand's address is incremented by 1 if the operand is a byte, or by 2 if the operand is a word.

**INDEXED (LONG AND SHORT)**—The instruction contains an 8-bit address field and either an 8-bit or a 16-bit displacement field. The 8-bit address field gives the word address of a word register in the Register File which contains a 16-bit base address. The 8- or 16-bit displacement field contains a signed displacement that will be added to the base address to produce the address of the operand. The operand can be anywhere in memory.

The 8096BH contains a Zero Register at word address 0000H (and which contains 0000H). This register is available for performing comparisons and for use as a base register in indexed addressing. This effectively provides direct addressing to all 64K of memory.

In the 8096BH, the Stack Pointer is at word address 0018H in the Register File. If the 8-bit address field contains 18H, the Stack Pointer becomes the base register. This allows direct accessing of variables in the stack.

The following tables list the MCS-96 instructions, their opcodes, and execution times.

# Instruction Summary

| Mnemonic        | Operands | Operation (Note 1)   | Flags |   |   |   |    |    | Notes |
|-----------------|----------|--|-------|---|---|---|----|----|-------|
|                 |          |  | Z     | N | C | V | VT | ST |       |
| ADD/ADDB        | 2        | $D \leftarrow D + A$   | ✓     | ✓ | ✓ | ✓ | ↑  | —  |       |
| ADD/ADDB        | 3        | $D \leftarrow B + A$   | ✓     | ✓ | ✓ | ✓ | ↑  | —  |       |
| ADDC/ADDCB      | 2        | $D \leftarrow D + A + C$   | ↓     | ✓ | ✓ | ✓ | ↑  | —  |       |
| SUB/SUBB        | 2        | $D \leftarrow D - A$   | ✓     | ✓ | ✓ | ✓ | ↑  | —  |       |
| SUB/SUBB        | 3        | $D \leftarrow B - A$   | ✓     | ✓ | ✓ | ✓ | ↑  | —  |       |
| SUBC/SUBCB      | 2        | $D \leftarrow D - A + C - 1$   | ↓     | ✓ | ✓ | ✓ | ↑  | —  |       |
| CMP/CMPB        | 2        | $D - A$  | ✓     | ✓ | ✓ | ✓ | ↑  | —  |       |
| MUL/MULU        | 2        | $D, D + 2 \leftarrow D * A$  | —     | — | — | — | —  | ?  | 2     |
| MUL/MULU        | 3        | $D, D + 2 \leftarrow B * A$  | —     | — | — | — | —  | ?  | 2     |
| MULB/MULUB      | 2        | $D, D + 1 \leftarrow D * A$  | —     | — | — | — | —  | ?  | 3     |
| MULB/MULUB      | 3        | $D, D + 1 \leftarrow B * A$  | —     | — | — | — | —  | ?  | 3     |
| DIVU            | 2        | $D \leftarrow (D, D + 2)/A, D + 2 \leftarrow \text{remainder}$                           | —     | — | — | ✓ | ↑  | —  | 2     |
| DIVUB           | 2        | $D \leftarrow (D, D + 1)/A, D + 1 \leftarrow \text{remainder}$                           | —     | — | — | ✓ | ↑  | —  | 3     |
| DIV             | 2        | $D \leftarrow (D, D + 2)/A, D + 2 \leftarrow \text{remainder}$                           | —     | — | — | ? | ↑  | —  |       |
| DIVB            | 2        | $D \leftarrow (D, D + 1)/A, D + 1 \leftarrow \text{remainder}$                           | —     | — | — | ? | ↑  | —  |       |
| AND/ANDB        | 2        | $D \leftarrow D \text{ and } A$  | ✓     | ✓ | 0 | 0 | —  | —  |       |
| AND/ANDB        | 3        | $D \leftarrow B \text{ and } A$  | ✓     | ✓ | 0 | 0 | —  | —  |       |
| OR/ORB          | 2        | $D \leftarrow D \text{ or } A$   | ✓     | ✓ | 0 | 0 | —  | —  |       |
| XOR/XORB        | 2        | $D \leftarrow D \text{ (excl. or) } A$   | ✓     | ✓ | 0 | 0 | —  | —  |       |
| LD/LDB          | 2        | $D \leftarrow A$   | —     | — | — | — | —  | —  |       |
| ST/STB          | 2        | $A \leftarrow D$   | —     | — | — | — | —  | —  |       |
| LDBSE           | 2        | $D \leftarrow A; D + 1 \leftarrow \text{SIGN}(A)$  | —     | — | — | — | —  | —  | 3, 4  |
| LDBZE           | 2        | $D \leftarrow A; D + 1 \leftarrow 0$   | —     | — | — | — | —  | —  | 3, 4  |
| PUSH            | 1        | $SP \leftarrow SP - 2; (SP) \leftarrow A$  | —     | — | — | — | —  | —  |       |
| POP             | 1        | $A \leftarrow (SP); SP \leftarrow SP + 2$  | —     | — | — | — | —  | —  |       |
| PUSHF           | 0        | $SP \leftarrow SP - 2; (SP) \leftarrow \text{PSW};$<br>$\text{PSW} \leftarrow 0000H$     | 0     | 0 | 0 | 0 | 0  | 0  |       |
| POPF            | 0        | $\text{PSW} \leftarrow (SP); SP \leftarrow SP + 2;$<br>$I \leftarrow 0$                  | ✓     | ✓ | ✓ | ✓ | ✓  | ✓  |       |
| SJMP            | 1        | $PC \leftarrow PC + 11\text{-bit offset}$  | —     | — | — | — | —  | —  | 5     |
| LJMP            | 1        | $PC \leftarrow PC + 16\text{-bit offset}$  | —     | — | — | — | —  | —  | 5     |
| BR [indirect]   | 1        | $PC \leftarrow (A)$  | —     | — | — | — | —  | —  |       |
| SCALL           | 1        | $SP \leftarrow SP - 2; (SP) \leftarrow PC;$<br>$PC \leftarrow PC + 11\text{-bit offset}$ | —     | — | — | — | —  | —  | 5     |
| LCALL           | 1        | $SP \leftarrow SP - 2; (SP) \leftarrow PC;$<br>$PC \leftarrow PC + 16\text{-bit offset}$ | —     | — | — | — | —  | —  | 5     |
| RET             | 0        | $PC \leftarrow (SP); SP \leftarrow SP + 2$   | —     | — | — | — | —  | —  |       |
| J (conditional) | 1        | $PC \leftarrow PC + 8\text{-bit offset (if taken)}$                                      | —     | — | — | — | —  | —  | 5     |
| JC              | 1        | Jump if C = 1  | —     | — | — | — | —  | —  | 5     |
| JNC             | 1        | Jump if C = 0  | —     | — | — | — | —  | —  | 5     |
| JE              | 1        | Jump if Z = 1  | —     | — | — | — | —  | —  | 5     |

## NOTES:

- If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
- D, D + 2 are consecutive WORDS in memory; D is DOUBLE-WORD aligned.
- D, D + 1 are consecutive BYTES in memory; D is WORD aligned.
- Changes a byte to a word.
- Offset is a 2's complement number.

## Instruction Summary (Continued)

| Mnemonic         | Oper-<br>ands | Operation (Note 1)                                 | Flags |   |   |   |    |    | Notes |
|------------------|---------------|--|-------|---|---|---|----|----|-------|
|                  |               |  | Z     | N | C | V | VT | ST |       |
| JNE              | 1             | Jump if Z = 0                                      | —     | — | — | — | —  | —  | 5     |
| JGE              | 1             | Jump if N = 0                                      | —     | — | — | — | —  | —  | 5     |
| JLT              | 1             | Jump if N = 1                                      | —     | — | — | — | —  | —  | 5     |
| JGT              | 1             | Jump if N = 0 and Z = 0                            | —     | — | — | — | —  | —  | 5     |
| JLE              | 1             | Jump if N = 1 or Z = 1                             | —     | — | — | — | —  | —  | 5     |
| JH               | 1             | Jump if C = 1 and Z = 0                            | —     | — | — | — | —  | —  | 5     |
| JNH              | 1             | Jump if C = 0 or Z = 1                             | —     | — | — | — | —  | —  | 5     |
| JV               | 1             | Jump if V = 1                                      | —     | — | — | — | —  | —  | 5     |
| JNV              | 1             | Jump if V = 0                                      | —     | — | — | — | —  | —  | 5     |
| JVT              | 1             | Jump if VT = 1; Clear VT                           | —     | — | — | — | 0  | —  | 5     |
| JNVT             | 1             | Jump if VT = 0; Clear VT                           | —     | — | — | — | 0  | —  | 5     |
| JST              | 1             | Jump if ST = 1                                     | —     | — | — | — | —  | —  | 5     |
| JNST             | 1             | Jump if ST = 0                                     | —     | — | — | — | —  | —  | 5     |
| JBS              | 3             | Jump if Specified Bit = 1                          | —     | — | — | — | —  | —  | 5, 6  |
| JBC              | 3             | Jump if Specified Bit = 0                          | —     | — | — | — | —  | —  | 5, 6  |
| DJNZ             | 1             | D ← D - 1; if D ≠ 0 then<br>PC ← PC + 8-bit offset | —     | — | — | — | —  | —  | 5     |
| DEC/DECB         | 1             | D ← D - 1  | ✓     | ✓ | ✓ | ✓ | ↑  | —  |       |
| NEG/NEGB         | 1             | D ← 0 - D  | ✓     | ✓ | ✓ | ✓ | ↑  | —  |       |
| INC/INCB         | 1             | D ← D + 1  | ✓     | ✓ | ✓ | ✓ | ↑  | —  |       |
| EXT              | 1             | D ← D; D + 2 ← Sign (D)                            | ✓     | ✓ | 0 | 0 | —  | —  | 2     |
| EXTB             | 1             | D ← D; D + 1 ← Sign(D)                             | ✓     | ✓ | 0 | 0 | —  | —  | 3     |
| NOT/NOTB         | 1             | D ← Logical Not (D)                                | ✓     | ✓ | 0 | 0 | —  | —  |       |
| CLR/CLRB         | 1             | D ← 0  | 1     | 0 | 0 | 0 | —  | —  |       |
| SHL/SHLB/SHLL    | 2             | C ← msb ———— lsb ← 0                               | ✓     | ? | ✓ | ✓ | ↑  | —  | 7     |
| SHR/SHRB/SHRL    | 2             | 0 → msb ———— lsb → C                               | ✓     | ? | ✓ | 0 | —  | ✓  | 7     |
| SHRA/SHRAB/SHRAL | 2             | msb → msb ———— lsb → C                             | ✓     | ✓ | ✓ | 0 | —  | ✓  | 7     |
| SETC             | 0             | C ← 1  | —     | — | 1 | — | —  | —  |       |
| CLRC             | 0             | C ← 0  | —     | — | 0 | — | —  | —  |       |
| CLRVT            | 0             | VT ← 0   | —     | — | — | — | 0  | —  |       |
| RST              | 0             | PC ← 2080H   | 0     | 0 | 0 | 0 | 0  | 0  | 8     |
| DI               | 0             | Disable All Interrupts (I ← 0)                     | —     | — | — | — | —  | —  |       |
| EI               | 0             | Enable All Interrupts (I ← 1)                      | —     | — | — | — | —  | —  |       |
| NOP              | 0             | PC ← PC + 1  | —     | — | — | — | —  | —  |       |
| SKIP             | 0             | PC ← PC + 2  | —     | — | — | — | —  | —  |       |
| NORML            | 2             | Left shift till msb = 1; D ← shift count           | ✓     | ? | 0 | — | —  | —  | 7     |
| TRAP             | 0             | SP ← SP - 2; (SP) ← PC<br>PC ← (2010H)             | —     | — | — | — | —  | —  | 9     |

## NOTES:

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
5. Offset is a 2's complement number.
6. Specified bit is one of the 2048 bits in the register file.
7. The "L" (Long) suffix indicates double-word operation.
8. Initiates a Reset by pulling RESET low. Software should re-initialize all the necessary registers with code starting at 2080H.
9. The assembler will not accept this mnemonic.



# Opcode and State Time Listing

| MNEMONIC                | OPERANDS | DIRECT |       |                | IMMEDIATE |       |                | INDIRECT <sup>Ⓢ</sup> |       |                             |       | INDEXED <sup>Ⓢ</sup>        |        |       |  |       |  |
|-------------------------|----------|--------|-------|----------------|-----------|-------|----------------|-----------------------|-------|-----------------------------|-------|-----------------------------|--------|-------|--|-------|--|
|                         |          | OPCODE | BYTES | STATE<br>TIMES | OPCODE    | BYTES | STATE<br>TIMES | NORMAL                |       | AUTO-INC.                   |       | SHORT                       |        | LONG  |  |       |  |
|                         |          |        |       |                |           |       |                | OPCODE                | BYTES | STATE <sup>①</sup><br>TIMES | BYTES | STATE <sup>①</sup><br>TIMES | OPCODE | BYTES | STATE <sup>①</sup><br>TIMES <sup>Ⓢ</sup> | BYTES | STATE <sup>①</sup><br>TIMES <sup>Ⓢ</sup> |
| ARITHMETIC INSTRUCTIONS |          |        |       |                |           |       |                |                       |       |                             |       |                             |        |       |  |       |  |
| ADD                     | 2        | 64     | 3     | 4              | 65        | 4     | 5              | 66                    | 3     | 6/11                        | 3     | 7/12                        | 67     | 4     | 6/11                                     | 5     | 7/12                                     |
| ADD                     | 3        | 44     | 4     | 5              | 45        | 5     | 6              | 46                    | 4     | 7/12                        | 4     | 8/13                        | 47     | 5     | 7/12                                     | 6     | 8/13                                     |
| ADDB                    | 2        | 74     | 3     | 4              | 75        | 3     | 4              | 76                    | 3     | 6/11                        | 3     | 7/12                        | 77     | 4     | 6/11                                     | 5     | 7/12                                     |
| ADDB                    | 3        | 54     | 4     | 5              | 55        | 4     | 5              | 56                    | 4     | 7/12                        | 4     | 8/13                        | 57     | 5     | 7/12                                     | 6     | 8/13                                     |
| ADDC                    | 2        | A4     | 3     | 4              | A5        | 4     | 5              | A6                    | 3     | 6/11                        | 3     | 7/12                        | A7     | 4     | 6/11                                     | 5     | 7/12                                     |
| ADDCB                   | 2        | B4     | 3     | 4              | B5        | 3     | 4              | B6                    | 3     | 6/11                        | 3     | 7/12                        | B7     | 4     | 6/11                                     | 5     | 7/12                                     |
| SUB                     | 2        | 68     | 3     | 4              | 69        | 4     | 5              | 6A                    | 3     | 6/11                        | 3     | 7/12                        | 6B     | 4     | 6/11                                     | 5     | 7/12                                     |
| SUB                     | 3        | 48     | 4     | 5              | 49        | 5     | 6              | 4A                    | 4     | 7/12                        | 4     | 8/13                        | 4B     | 5     | 7/12                                     | 6     | 8/13                                     |
| SUBB                    | 2        | 78     | 3     | 4              | 79        | 3     | 4              | 7A                    | 3     | 6/11                        | 3     | 7/12                        | 7B     | 4     | 6/11                                     | 5     | 7/12                                     |
| SUBB                    | 3        | 58     | 4     | 5              | 59        | 4     | 5              | 5A                    | 4     | 7/12                        | 4     | 8/13                        | 5B     | 5     | 7/12                                     | 6     | 8/13                                     |
| SUBC                    | 2        | A8     | 3     | 4              | A9        | 4     | 5              | AA                    | 3     | 6/11                        | 3     | 7/12                        | AB     | 4     | 6/11                                     | 5     | 7/12                                     |
| SUBCB                   | 2        | B8     | 3     | 4              | B9        | 3     | 4              | BA                    | 3     | 6/11                        | 3     | 7/12                        | BB     | 4     | 6/11                                     | 5     | 7/12                                     |
| CMP                     | 2        | 88     | 3     | 4              | 89        | 4     | 5              | 8A                    | 3     | 6/11                        | 3     | 7/12                        | 8B     | 4     | 6/11                                     | 5     | 7/12                                     |
| CMPB                    | 2        | 98     | 3     | 4              | 99        | 3     | 4              | 9A                    | 3     | 6/11                        | 3     | 7/12                        | 9B     | 4     | 6/11                                     | 5     | 7/12                                     |
|                         |          |        |       |                |           |       |                |                       |       |                             |       |                             |        |       |  |       |  |
| MULU                    | 2        | 6C     | 3     | 25             | 6D        | 4     | 26             | 6E                    | 3     | 27/32                       | 3     | 28/33                       | 6F     | 4     | 27/32                                    | 5     | 28/33                                    |
| MULU                    | 3        | 4C     | 4     | 26             | 4D        | 5     | 27             | 4E                    | 4     | 28/33                       | 4     | 29/34                       | 4F     | 5     | 28/33                                    | 6     | 29/34                                    |
| MULUB                   | 2        | 7C     | 3     | 17             | 7D        | 3     | 17             | 7E                    | 3     | 19/24                       | 3     | 20/25                       | 7F     | 4     | 19/24                                    | 5     | 20/25                                    |
| MULUB                   | 3        | 5C     | 4     | 18             | 5D        | 4     | 18             | 5E                    | 4     | 20/25                       | 4     | 21/26                       | 5F     | 5     | 20/25                                    | 6     | 21/26                                    |
| MUL                     | 2        | ②      | 4     | 29             | ②         | 5     | 30             | ②                     | 4     | 31/36                       | 4     | 32/37                       | ②      | 5     | 31/36                                    | 6     | 32/37                                    |
| MUL                     | 3        | ②      | 5     | 30             | ②         | 6     | 31             | ②                     | 5     | 32/37                       | 5     | 33/38                       | ②      | 6     | 32/37                                    | 7     | 33/38                                    |
| MULB                    | 2        | ②      | 4     | 21             | ②         | 4     | 21             | ②                     | 4     | 23/28                       | 4     | 24/29                       | ②      | 5     | 23/28                                    | 6     | 24/29                                    |
| MULB                    | 3        | ②      | 5     | 22             | ②         | 5     | 22             | ②                     | 5     | 24/29                       | 5     | 25/30                       | ②      | 6     | 24/29                                    | 7     | 25/30                                    |
| DIVU                    | 2        | 8C     | 3     | 25             | 8D        | 4     | 26             | 8E                    | 3     | 28/32                       | 3     | 29/33                       | 8F     | 4     | 28/32                                    | 5     | 29/33                                    |
| DIVUB                   | 2        | 9C     | 3     | 17             | 9D        | 3     | 17             | 9E                    | 3     | 20/24                       | 3     | 21/25                       | 9F     | 4     | 20/24                                    | 5     | 21/25                                    |
| DIV                     | 2        | ②      | 4     | 29             | ②         | 5     | 30             | ②                     | 4     | 32/36                       | 4     | 33/37                       | ②      | 5     | 32/36                                    | 6     | 33/37                                    |
| DIVB                    | 2        | ②      | 4     | 21             | ②         | 4     | 21             | ②                     | 4     | 24/28                       | 4     | 25/29                       | ②      | 5     | 24/28                                    | 6     | 25/29                                    |

## Notes:

Ⓢ Long indexed and Indirect+ instructions have identical opcodes with Short indexed and Indirect modes, respectively. The second byte of instructions using any indirect or indexed addressing mode specifies the exact mode used. If the second byte is even, use Indirect or Short Indexed. If it is odd, use Indirect+ or Long indexed. In all cases the second byte of the instruction always specifies an even (word) location for the address referenced.

① Number of state times shown for internal/external operands.

② The opcodes for signed multiply and divide are the opcodes for the unsigned functions with an 'FE' appended as a prefix.

Ⓢ State times shown for 16-bit bus.

# Opcode and State Time Listing (Continued)

| MNEMONIC                          | OPERANDS           | DIRECT |       |                | IMMEDIATE |       |                | INDIRECT <sup>①</sup> |                   |                             |       | INDEXED <sup>②</sup>        |        |       |  |       |  |
|-----------------------------------|--------------------|--------|-------|----------------|-----------|-------|----------------|-----------------------|-------------------|-----------------------------|-------|-----------------------------|--------|-------|--|-------|--|
|                                   |                    | OPCODE | BYTES | STATE<br>TIMES | OPCODE    | BYTES | STATE<br>TIMES | NORMAL                |                   | AUTO-INC.                   |       | SHORT                       |        | LONG  |  |       |  |
|                                   |                    |        |       |                |           |       |                | OPCODE                | BYTES             | STATE <sup>①</sup><br>TIMES | BYTES | STATE <sup>①</sup><br>TIMES | OPCODE | BYTES | STATE <sup>①</sup><br>TIMES <sup>③</sup> | BYTES | STATE <sup>①</sup><br>TIMES <sup>③</sup> |
|                                   |                    |        |       |                |           |       |                |                       |                   |                             |       |                             |        |       |  |       |  |
| LOGICAL INSTRUCTIONS              |                    |        |       |                |           |       |                |                       |                   |                             |       |                             |        |       |  |       |  |
| AND                               | 2                  | 60     | 3     | 4              | 61        | 4     | 5              | 62                    | 3                 | 6/11                        | 3     | 7/12                        | 63     | 4     | 6/11                                     | 5     | 7/12                                     |
| AND                               | 3                  | 40     | 4     | 5              | 41        | 5     | 6              | 42                    | 4                 | 7/12                        | 4     | 8/13                        | 43     | 5     | 7/12                                     | 6     | 8/13                                     |
| ANDB                              | 2                  | 70     | 3     | 4              | 71        | 3     | 4              | 72                    | 3                 | 6/11                        | 3     | 7/12                        | 73     | 4     | 6/11                                     | 5     | 7/12                                     |
| ANDB                              | 3                  | 50     | 4     | 5              | 51        | 4     | 5              | 52                    | 4                 | 7/12                        | 4     | 8/13                        | 53     | 5     | 7/12                                     | 6     | 8/13                                     |
| OR                                | 2                  | 80     | 3     | 4              | 81        | 4     | 5              | 82                    | 3                 | 6/11                        | 3     | 7/12                        | 83     | 4     | 6/11                                     | 5     | 7/12                                     |
| ORB                               | 2                  | 90     | 3     | 4              | 91        | 3     | 4              | 92                    | 3                 | 6/11                        | 3     | 7/12                        | 93     | 4     | 6/11                                     | 5     | 7/12                                     |
| XOR                               | 2                  | 84     | 3     | 4              | 85        | 4     | 5              | 86                    | 3                 | 6/11                        | 3     | 7/12                        | 87     | 4     | 6/11                                     | 5     | 7/12                                     |
| XORB                              | 2                  | 94     | 3     | 4              | 95        | 3     | 4              | 96                    | 3                 | 6/11                        | 3     | 7/12                        | 97     | 4     | 6/11                                     | 5     | 7/12                                     |
| DATA TRANSFER INSTRUCTIONS        |                    |        |       |                |           |       |                |                       |                   |                             |       |                             |        |       |  |       |  |
| LD                                | 2                  | A0     | 3     | 4              | A1        | 4     | 5              | A2                    | 3                 | 6/11                        | 3     | 7/12                        | A3     | 4     | 6/11                                     | 5     | 7/12                                     |
| LDB                               | 2                  | B0     | 3     | 4              | B1        | 3     | 4              | B2                    | 3                 | 6/11                        | 3     | 7/12                        | B3     | 4     | 6/11                                     | 5     | 7/12                                     |
| ST                                | 2                  | C0     | 3     | 4              | —         | —     | —              | C2                    | 3                 | 7/11                        | 3     | 8/12                        | C3     | 4     | 7/11                                     | 5     | 8/12                                     |
| STB                               | 2                  | C4     | 3     | 4              | —         | —     | —              | C6                    | 3                 | 7/11                        | 3     | 8/12                        | C7     | 4     | 7/11                                     | 5     | 8/12                                     |
| LDBSE                             | 2                  | BC     | 3     | 4              | BD        | 3     | 4              | BE                    | 3                 | 6/11                        | 3     | 7/12                        | BF     | 4     | 6/11                                     | 5     | 7/12                                     |
| LDBZE                             | 2                  | AC     | 3     | 4              | AD        | 3     | 4              | AE                    | 3                 | 6/11                        | 3     | 7/12                        | AF     | 4     | 6/11                                     | 5     | 7/12                                     |
| STACK OPERATIONS (internal stack) |                    |        |       |                |           |       |                |                       |                   |                             |       |                             |        |       |  |       |  |
| PUSH                              | 1                  | C8     | 2     | 8              | C9        | 3     | 8              | CA                    | 2                 | 11/15                       | 2     | 12/16                       | CB     | 3     | 11/15                                    | 4     | 12/16                                    |
| POP                               | 1                  | CC     | 2     | 12             | —         | —     | —              | CE                    | 2                 | 14/18                       | 2     | 14/18                       | CF     | 3     | 14/18                                    | 4     | 14/18                                    |
| PUSHF                             | 0                  | F2     | 1     | 8              | —         | —     | —              | —                     | —                 | —                           | —     | —                           | —      | —     | —  | —     | —  |
| POPF                              | 0                  | F3     | 1     | 9              | —         | —     | —              | —                     | —                 | —                           | —     | —                           | —      | —     | —  | —     | —  |
| STACK OPERATIONS (external stack) |                    |        |       |                |           |       |                |                       |                   |                             |       |                             |        |       |  |       |  |
| PUSH                              | 1                  | C8     | 2     | 12             | C9        | 3     | 12             | CA                    | 2                 | 15/19                       | 2     | 16/20                       | CB     | 3     | 15/19                                    | 4     | 16/20                                    |
| POP                               | 1                  | CC     | 2     | 14             | —         | —     | —              | CE                    | 2                 | 16/20                       | 2     | 16/20                       | CF     | 3     | 16/20                                    | 4     | 16/20                                    |
| PUSHF                             | 0                  | F2     | 1     | 12             | —         | —     | —              | —                     | —                 | —                           | —     | —                           | —      | —     | —  | —     | —  |
| POPF                              | 0                  | F3     | 1     | 13             | —         | —     | —              | —                     | —                 | —                           | —     | —                           | —      | —     | —  | —     | —  |
| JUMPS AND CALLS                   |                    |        |       |                |           |       |                |                       |                   |                             |       |                             |        |       |  |       |  |
| MNEMONIC                          | OPCODE             |        | BYTES |                | STATES    |       | MNEMONIC       | OPCODE                |                   | BYTES                       |       | STATES                      |        |       |  |       |  |
| LJMP                              | E7                 |        | 3     |                | 8         |       | LCALL          | EF                    |                   | 3                           |       | 13/16 <sup>⑤</sup>          |        |       |  |       |  |
| SJMP                              | 20-27 <sup>④</sup> |        | 2     |                | 8         |       | SCALL          | 28-2F <sup>④</sup>    |                   | 2                           |       | 13/16 <sup>⑤</sup>          |        |       |  |       |  |
| BR[ ]                             | E3                 |        | 2     |                | 8         |       | RET            | F0                    |                   | 1                           |       | 12/16 <sup>⑤</sup>          |        |       |  |       |  |
| Notes:                            |                    |        |       |                |           |       |                |                       | TRAP <sup>③</sup> |                             | F7    |                             | 1      |       | 21/24                                    |       |  |

## Notes:

- ① Number of state times shown for internal/external operands.
- ② The assembler does not accept this mnemonic.
- ③ The least significant 3 bits of the opcode are concatenated with the following 8 bits to form an 11-bit, 2's complement, offset for the relative call or jump.
- ④ State times for stack located internal/external.
- ⑤ State times shown for 16-bit bus.

### CONDITIONAL JUMPS

| All conditional jumps are 2 byte instructions. They require 8 state times if the jump is taken, 4 if it is not. <sup>(8)</sup> |        |          |        |          |        |          |        |
|--|--------|----------|--------|----------|--------|----------|--------|
| MNEMONIC   | OPCODE | MNEMONIC | OPCODE | MNEMONIC | OPCODE | MNEMONIC | OPCODE |
| JC   | DB     | JE       | DF     | JGE      | D6     | JGT      | D2     |
| JNC  | D3     | JNE      | D7     | JLT      | DE     | JLE      | DA     |
| JH   | D9     | JV       | DD     | JVT      | DC     | JST      | D8     |
| JNH  | D1     | JNV      | D5     | JNVT     | D4     | JNST     | D0     |

### JUMP ON BIT CLEAR OR BIT SET

| These instructions are 3-byte instructions. They require 9 state times if the jump is taken, 5 if it is not. <sup>(8)</sup> |    |    |    |    |    |    |    |    |
|---|----|----|----|----|----|----|----|----|
| BIT NUMBER  |    |    |    |    |    |    |    |    |
| MNEMONIC  | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  |
| JBC   | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 |
| JBS   | 38 | 39 | 3A | 3B | 3C | 3D | 3E | 3F |

### LOOP CONTROL

| MNEMONIC | OPCODE | BYTES | STATE TIMES                                     |
|----------|--------|-------|---|
| DJNZ     | EO     | 3     | 5/9 STATE TIME (NOT TAKEN/TAKEN) <sup>(8)</sup> |

### SINGLE REGISTER INSTRUCTIONS

| MNEMONIC | OPCODE | BYTES | STATES <sup>(8)</sup> | MNEMONIC | OPCODE | BYTES | STATES <sup>(8)</sup> |
|----------|--------|-------|-----------------------|----------|--------|-------|-----------------------|
| DEC      | 05     | 2     | 4                     | EXT      | 06     | 2     | 4                     |
| DECB     | 15     | 2     | 4                     | EXTB     | 16     | 2     | 4                     |
| NEG      | 03     | 2     | 4                     | NOT      | 02     | 2     | 4                     |
| NEGB     | 13     | 2     | 4                     | NOTB     | 12     | 2     | 4                     |
| INC      | 07     | 2     | 4                     | CLR      | 01     | 2     | 4                     |
| INCB     | 17     | 2     | 4                     | CLRB     | 11     | 2     | 4                     |

### SHIFT INSTRUCTIONS

| INSTR<br>MNEMONIC | WORD |   | INSTR<br>MNEMONIC | BYTE |   | INSTR<br>MNEMONIC | DBL WD |   | STATE TIMES <sup>(8)</sup>     |
|-------------------|------|---|-------------------|------|---|-------------------|--------|---|--------------------------------|
|                   | OP   | B |                   | OP   | B |                   | OP     | B |                                |
| SHL               | 09   | 3 | SHLB              | 19   | 3 | SHLL              | 0D     | 3 | 7 + 1 PER SHIFT <sup>(7)</sup> |
| SHR               | 08   | 3 | SHRB              | 18   | 3 | SHRL              | 0C     | 3 | 7 + 1 PER SHIFT <sup>(7)</sup> |
| SHRA              | 0A   | 3 | SHRAB             | 1A   | 3 | SHRAL             | 0E     | 3 | 7 + 1 PER SHIFT <sup>(7)</sup> |

### SPECIAL CONTROL INSTRUCTIONS

| MNEMONIC | OPCODE | BYTES | STATES <sup>(8)</sup> | MNEMONIC | OPCODE | BYTES | STATES <sup>(8)</sup> |
|----------|--------|-------|-----------------------|----------|--------|-------|-----------------------|
| SETC     | F9     | 1     | 4                     | DI       | FA     | 1     | 4                     |
| CLRC     | F8     | 1     | 4                     | EI       | FB     | 1     | 4                     |
| CLRVT    | FC     | 1     | 4                     | NOP      | FD     | 1     | 4                     |
| RST      | FF     | 1     | 166                   | SKIP     | 00     | 2     | 4                     |

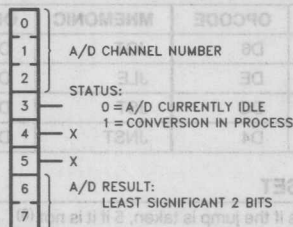
### NORMALIZE

| MNEMONIC | OPCODE | BYTES | STATE TIMES      |
|----------|--------|-------|------------------|
| NORML    | 0F     | 3     | 11 + 1 PER SHIFT |

#### NOTES:

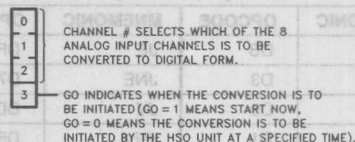
- This instruction takes 2 states to pull RST low, then holds it low for 2 states to initiate a reset. The reset takes 12 states, at which time the program restarts at location 2080H.
- Execution will take at least 8 states, even for 0 shift.
- State times shown for 16-bit bus.

### A/D Result LO (02H)



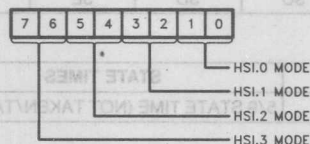
270090-21

### A/D Command (02H)



270090-24

### HSI\_Mode (03H)

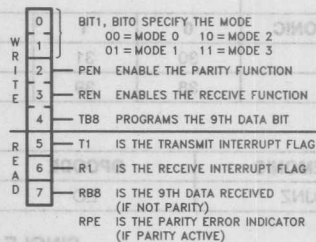


WHERE EACH 2-BIT MODE CONTROL FIELD  
DEFINES ONE OF 4 POSSIBLE MODES:

- 00 8 POSITIVE TRANSITIONS
- 01 EACH POSITIVE TRANSITION
- 10 EACH NEGATIVE TRANSITION
- 11 EVERY TRANSITION (POSITIVE AND NEGATIVE)

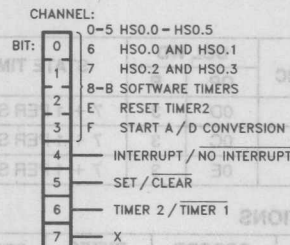
270090-22

### SPCON/SPSTAT (11H)



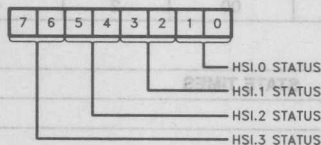
270090-26

### HSO Command (06H)



270090-23

### HSI\_Status (06H)



WHERE FOR EACH 2-BIT STATUS FIELD THE LOWER BIT INDICATES WHETHER OR NOT AN EVENT HAS OCCURRED ON THIS PIN AND THE UPPER BIT INDICATES THE CURRENT STATUS OF THE PIN.

270090-25

### Baud Rate Calculations

Using XTAL1:

$$\text{Mode 0: Baud Rate} = \frac{\text{XTAL1 frequency}}{4 \cdot (B + 1)}; B \neq 0$$

$$\text{Others: Baud Rate} = \frac{\text{XTAL1 frequency}}{64 \cdot (B + 1)}$$

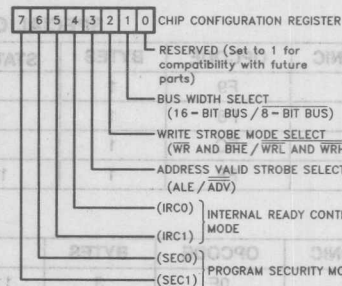
Using T2CLK:

$$\text{Mode 0: Baud Rate} = \frac{\text{T2CLK frequency}}{B}; B \neq 0$$

$$\text{Others: Baud Rate} = \frac{\text{T2CLK frequency}}{16 \cdot B}; B \neq 0$$

Note that B cannot equal 0, except when using XTAL1 in other than mode 0.

### Chip Configuration



270090-32

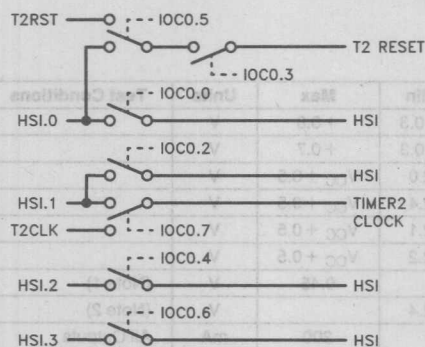


### IOC0 (15H)

- 0 — HSI.0 INPUT ENABLE / DISABLE
- 1 — TIMER 2 RESET EACH WRITE
- 2 — HSI.1 INPUT ENABLE / DISABLE
- 3 — TIMER 2 EXTERNAL RESET ENABLE / DISABLE
- 4 — HSI.2 INPUT ENABLE / DISABLE
- 5 — TIMER 2 RESET SOURCE HSI.0 / T2RST
- 6 — HSI.3 INPUT ENABLE / DISABLE
- 7 — TIMER 2 CLOCK SOURCE HSI.1 / T2CLK

270090-30

### IOC0 (15H)



270090-29

### IOS0 (15H)

- 0 — HSO.0 CURRENT STATE
- 1 — HSO.1 CURRENT STATE
- 2 — HSO.2 CURRENT STATE
- 3 — HSO.3 CURRENT STATE
- 4 — HSO.4 CURRENT STATE
- 5 — HSO.5 CURRENT STATE
- 6 — CAM OR HOLDING REGISTER IS FULL
- 7 — HSO HOLDING REGISTER IS FULL

270090-27

### IOC1 (16H)

- 0 — SELECT PWM / SELECT P2.5
- 1 — EXTERNAL INTERRUPT ACH7 / EXTINT
- 2 — TIMER 1 OVERFLOW INTERRUPT ENABLE / DISABLE
- 3 — TIMER 2 OVERFLOW INTERRUPT ENABLE / DISABLE
- 4 — HSO.4 OUTPUT ENABLE / DISABLE
- 5 — SELECT TXD / SELECT P2.0
- 6 — HSO.5 OUTPUT ENABLE / DISABLE
- 7 — HSI INTERRUPT  
FIFO FULL / HOLDING REGISTER LOADED

270090-31

| Vector         | Vector Location |            | Priority       |
|----------------|-----------------|------------|----------------|
|                | (High Byte)     | (Low Byte) |                |
| Software       | 2011H           | 2010H      | Not Applicable |
| Extint         | 200FH           | 200EH      |                |
| Serial Port    | 200DH           | 200CH      | 6              |
| Software       | 200BH           | 200AH      | 5              |
| Timers         |                 |            |                |
| HSI.0          | 2009H           | 2008H      | 4              |
| High Speed     | 2007H           | 2006H      | 3              |
| Outputs        |                 |            |                |
| HSI Data       | 2005H           | 2004H      | 2              |
| Available      |                 |            |                |
| A/D Conversion | 2003H           | 2002H      | 1              |
| Complete       |                 |            |                |
| Timer Overflow | 2001H           | 2000H      | 0 (Lowest)     |

### IOS1 (16H)

- 0 — SOFTWARE TIMER 0 EXPIRED
- 1 — SOFTWARE TIMER 1 EXPIRED
- 2 — SOFTWARE TIMER 2 EXPIRED
- 3 — SOFTWARE TIMER 3 EXPIRED
- 4 — TIMER 2 HAS OVERFLOW
- 5 — TIMER 1 HAS OVERFLOW
- 6 — HSI FIFO IS FULL
- 7 — HSI HOLDING REGISTER DATA AVAILABLE

270090-28

## ELECTRICAL CHARACTERISTICS ABSOLUTE MAXIMUM RATINGS\*

Ambient Temperature Under Bias . . . . 0°C to +70°C  
Storage Temperature . . . . . -40°C to +150°C  
Voltage from Any Pin to  
V<sub>SS</sub> or ANGND . . . . . -0.3V to +7.0V  
Average Output Current from Any Pin . . . . . 10 mA  
Power Dissipation . . . . . 1.5W

\*Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

NOTICE: Specifications contained within the following tables are subject to change.

## OPERATING CONDITIONS

| Symbol           | Parameter                      | Min  | Max  | Units |
|------------------|--------------------------------|------|------|-------|
| T <sub>A</sub>   | Ambient Temperature Under Bias | 0    | +70  | C     |
| V <sub>CC</sub>  | Digital Supply Voltage         | 4.50 | 5.50 | V     |
| V <sub>REF</sub> | Analog Supply Voltage          | 4.50 | 5.50 | V     |
| f <sub>OSC</sub> | Oscillator Frequency           | 6.0  | 12   | MHz   |
| V <sub>PD</sub>  | Power-Down Supply Voltage      | 4.50 | 5.50 | V     |

NOTE: ANGND and V<sub>SS</sub> should be nominally at the same potential.

## D.C. CHARACTERISTICS

| Symbol           | Parameter   | Min  | Max                   | Units | Test Conditions                                 |
|------------------|---|------|-----------------------|-------|---|
| V <sub>IL</sub>  | Input Low Voltage (Except RESET)                                  | -0.3 | +0.8                  | V     |   |
| V <sub>IL1</sub> | Input Low Voltage, RESET  | -0.3 | +0.7                  | V     |   |
| V <sub>IH</sub>  | Input High Voltage (Except RESET, NMI, XTAL1)                     | 2.0  | V <sub>CC</sub> + 0.5 | V     |   |
| V <sub>IH1</sub> | Input High Voltage, RESET Rising                                  | 2.4  | V <sub>CC</sub> + 0.5 | V     |   |
| V <sub>IH2</sub> | Input High Voltage, RESET Falling                                 | 2.1  | V <sub>CC</sub> + 0.5 | V     |   |
| V <sub>IH3</sub> | Input High Voltage, NMI, XTAL1                                    | 2.2  | V <sub>CC</sub> + 0.5 | V     |   |
| V <sub>OL</sub>  | Output Low Voltage  |      | 0.45                  | V     | (Note 1)  |
| V <sub>OH</sub>  | Output High Voltage   | 2.4  |                       | V     | (Note 2)  |
| I <sub>CC</sub>  | V <sub>CC</sub> Supply Current                                    |      | 200                   | mA    | All Outputs Disconnected.                       |
| I <sub>PD</sub>  | V <sub>PD</sub> Supply Current                                    |      | 1                     | mA    | Normal operation and Power-Down.                |
| I <sub>REF</sub> | V <sub>REF</sub> Supply Current                                   |      | 8                     | mA    |   |
| I <sub>LI</sub>  | Input Leakage Current to all pins of HSI, P0 P3, P4, and to P2.1. |      | ±10                   | μA    | V <sub>IN</sub> = 0 to V <sub>CC</sub> (Note 3) |
| I <sub>IH</sub>  | Input High Current to EA  |      | 100                   | μA    | V <sub>IH</sub> = 2.4V                          |
| I <sub>IL</sub>  | Input Low Current to all pins of P1, and to P2.6, P2.7.           |      | -100                  | μA    | V <sub>IL</sub> = 0.45V                         |
| I <sub>IL1</sub> | Input Low Current to RESET  |      | -2                    | mA    | V <sub>IL</sub> = 0.45V                         |
| I <sub>IL2</sub> | Input Low Current P2.2, P2.3, P2.4 READY                          |      | -50                   | μA    | V <sub>IL</sub> = 0.45V                         |
| C <sub>S</sub>   | Pin Capacitance (Any Pin to V <sub>SS</sub> )                     |      | 10                    | pF    | t <sub>TEST</sub> = 1.0 MHz                     |

### NOTES:

1. I<sub>OL</sub> = 0.36 mA for all pins of P1, for P2.6 and P2.7, and for all pins of P3 and P4 when used as ports.  
I<sub>OL</sub> = 2.0 mA for TXD, RXD (in serial port mode 0), PWM, CLKOUT, ALE, BHE, RD, WR, and all pins of HSO and P3 and P4 when used as external memory bus (AD0-AD15).
  2. I<sub>OH</sub> = -20 μA for all pins of P1, or P2.6 and P2.7.  
I<sub>OH</sub> = -200 μA for TXD, RXD (in serial port mode 0), PWM, CLKOUT, ALE, BHE, WR, and all pins of HSO and P3 and P4 when used as external memory bus (AD0-AD15).
  3. P3 and P4, when used as ports, have open-drain outputs.
2. Analog Conversion not in process.

## A/D CONVERTER SPECIFICATIONS

A/D Converter operation is verified only on the 8097BH, 8397BH, 8095BH, 8395BH, 8797BH, 8795BH.

The absolute conversion accuracy is dependent on the accuracy of  $V_{REF}$ . The specifications given below assume adherence to the Operating Conditions section of these data sheets. Testing is done at  $V_{REF} = 5.120V$ .

|                             |                                 |
|-----------------------------|---------------------------------|
| Resolution                  | $\pm 0.001 V_{REF}$             |
| Accuracy                    | $\pm 0.004 V_{REF}$             |
| Differential nonlinearity   | $\pm 0.002 V_{REF} \text{ max}$ |
| Integral nonlinearity       | $\pm 0.004 V_{REF} \text{ max}$ |
| Channel-to-channel matching | $\pm 1 \text{ LSB}$             |
| Crosstalk (DC to 100 KHz)   | $-60 \text{ dB max}$            |

A.C. CHARACTERISTICS ( $V_{CC}, V_{PD} = 4.5 \text{ to } 5.5V$ ;  $T_A = 0^\circ\text{C to } 70^\circ\text{C}$ ;  $f_{OSC} = 6.0 \text{ to } 12.0 \text{ MHz}$ )

Test Conditions: Load Capacitance on Output Pins = 80 pF

Oscillator Frequency = 12 MHz

## TIMING REQUIREMENTS (Other system components must meet these specs.)

| Symbol               | Parameter   | Min              | Max         | Units |
|----------------------|---|------------------|-------------|-------|
| TCLYX <sup>(5)</sup> | READY Hold after CLKOUT Edge                                | 0 <sup>(1)</sup> |             | ns    |
| TLLYV                | End of ALE to READY Setup                                   | -Tosc            | 2Tosc - 60  | ns    |
| TLLYH                | End of ALE to READY High                                    | 2Tosc + 40       | 4Tosc - 40  | ns    |
| TYLYH                | Non-ready Time  |                  | 1000        | ns    |
| TAVDV                | Address Valid to Input Data Valid                           |                  | 5Tosc - 80  | ns    |
| TRLDV                | $\overline{RD}$ Active to Input Data Valid                  |                  | 3Tosc - 60  | ns    |
| TRHDX                | Data Hold after $\overline{RD}$ Inactive <sup>(3)</sup>     | 0                |             | ns    |
| TRHDZ                | $\overline{RD}$ Inactive to Input Data Float <sup>(3)</sup> | 0                | Tosc - 20   | ns    |
| TAVGV <sup>(5)</sup> | Address valid to BUSWIDTH valid                             |                  | 2Tosc - 100 | ns    |
| TLLGX <sup>(5)</sup> | BUSWIDTH hold after ALE/ $\overline{ADV}$ low               | Tosc + 10        |             | ns    |
| TLLGV <sup>(5)</sup> | ALE/ $\overline{ADV}$ low to BUSWIDTH valid                 |                  | Tosc - 50   | ns    |

## NOTE:

5. Pins not bonded out on 48-pin parts.

## A.C. CHARACTERISTICS (Continued)

## TIMING RESPONSES (MCS-96 parts meet these specs.)

| Symbol               | Parameter  | Min                  | Max                  | Units |
|----------------------|--|----------------------|----------------------|-------|
| FXTAL                | Oscillator Frequency   | 6.0                  | 12.0                 | MHz   |
| Tosc                 | Oscillator Period  | 83                   | 166                  | ns    |
| TCHCH <sup>(5)</sup> | CLKOUT Period <sup>(3)</sup>                                     | 3Tosc <sup>(4)</sup> | 3Tosc <sup>(4)</sup> | ns    |
| TCHCL <sup>(5)</sup> | CLKOUT High Time   | Tosc - 20            | Tosc + 20            | ns    |
| TCLLH <sup>(5)</sup> | CLKOUT Low to ALE High   | - 5                  | 20                   | ns    |
| TLLCH <sup>(5)</sup> | ALE Low to CLKOUT High   | Tosc - 20            | Tosc + 40            | ns    |
| TLHLL                | ALE Pulse Width  | Tosc - 25            | Tosc + 15            | ns    |
| TAVLL                | Address Setup to End of ALE                                      | Tosc - 50            |                      | ns    |
| TLLRL                | End of ALE to $\overline{RD}$ or $\overline{WR}$ Active          | Tosc - 20            |                      | ns    |
| TLLAX                | Address hold after End of ALE                                    | Tosc - 20            |                      | ns    |
| TWLWH                | $\overline{WR}$ Pulse Width                                      | 3Tosc - 35           |                      | ns    |
| TQVWH                | Output Data Valid to End of $\overline{WR}$                      | 3Tosc - 60           |                      | ns    |
| TWHQX                | Output Data Hold after $\overline{WR}$                           | Tosc - 25            |                      | ns    |
| TWHLH                | End of $\overline{WR}$ to Next ALE                               | Tosc - 25            |                      | ns    |
| TRLRH                | $\overline{RD}$ Pulse Width                                      | 3Tosc - 30           |                      | ns    |
| TRHLH                | End of $\overline{RD}$ to Next ALE                               | Tosc - 25            |                      | ns    |
| TCLVL <sup>(5)</sup> | CLOCKOUT Low to $\overline{ADV}$ Low                             | Tosc - 20            | Tosc + 20            | ns    |
| TRHBX <sup>(5)</sup> | $\overline{RD}$ High to INST, $\overline{BHE}$ , AD8-15 Inactive | Tosc                 | Tosc + 30            | ns    |
| TWHBX <sup>(5)</sup> | $\overline{WR}$ High to INST, $\overline{BHE}$ , AD8-15 Inactive | Tosc                 | Tosc + 30            | ns    |
| THLHH                | WRL, WRH Low to WRL, WRH High                                    | 2Tosc - 20           | 2Tosc + 20           | ns    |
| TLLHL                | ALE Low to WRL, WRH Low  | 2Tosc - 20           | 2Tosc + 20           | ns    |
| TQVHL                | Output Data Valid to WRL, WRH Low                                | Tosc - 60            |                      | ns    |

## NOTES:

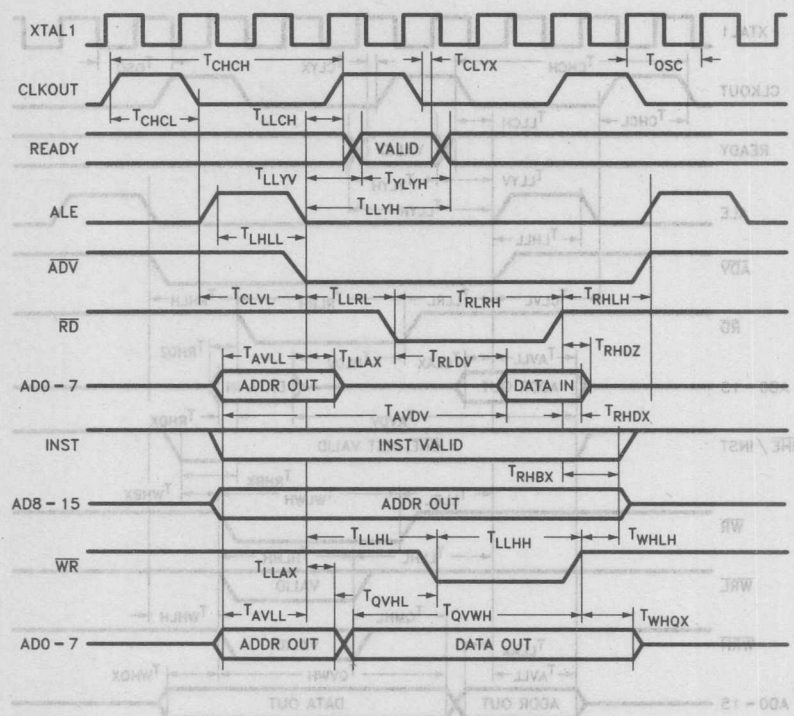
1. If the 48-pin part is being used then this timing can be generated by assuming that the CLKOUT falling edge has occurred at  $2Tosc + 60$  (TLLCH(max) + TCHCL(max)) after the falling edge of ALE.
2. If more than one wait state is desired, add 3Tosc for each additional wait state.
3. This specification is not tested, but is verified by design analysis and/or derived from other tested parameters.
4. CLKOUT is directly generated as a divide by 3 of the oscillator. The period will be  $3Tosc \pm 10$  ns if Tosc is constant and the rise and fall times on XTAL 1 are less than 10 ns.
5. Pins not bonded out on 48-pin parts.





# WAVEFORM — 8-BIT BUS

WAVEFORM — 8-BIT BUS



270090-34

## NOTE:

For word reads or writes in the 8-bit bus mode the 8X9XBH performs two complete bus cycles.

## A.C. CHARACTERISTICS—SERIAL PORT—SHIFT REGISTER MODE

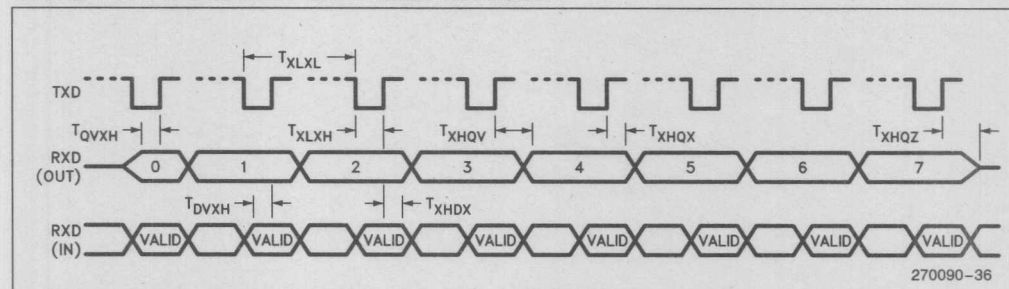
## SERIAL PORT TIMING—SHIFT REGISTER MODE

Test Conditions:  $T_A = 0^\circ\text{C}$  to  $+70^\circ\text{C}$ ;  $V_{CC} = 5\text{V} \pm 10\%$ ;  $V_{SS} = 0\text{V}$ ; Load Capacitance = 80 pF

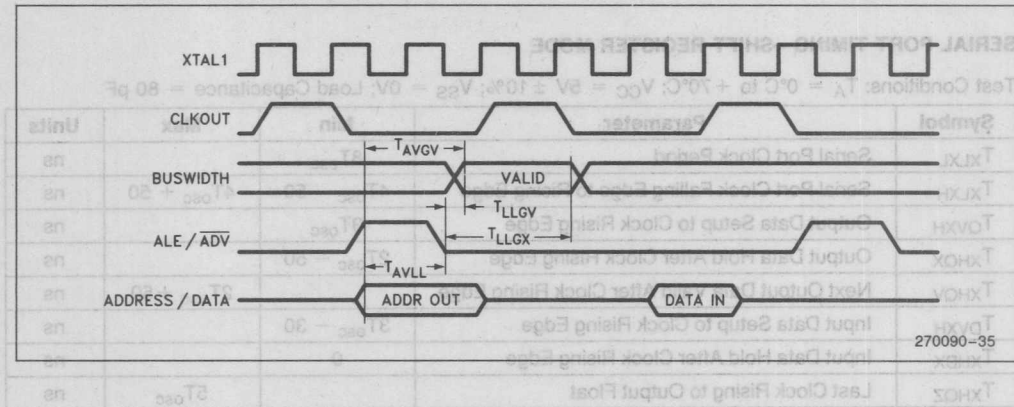
| Symbol     | Parameter                                      | Min             | Max             | Units |
|------------|--|-----------------|-----------------|-------|
| $T_{XLXL}$ | Serial Port Clock Period                       | $8T_{osc}$      |                 | ns    |
| $T_{XLXH}$ | Serial Port Clock Falling Edge to Rising Edge  | $4T_{osc} - 50$ | $4T_{osc} + 50$ | ns    |
| $T_{QVXH}$ | Output Data Setup to Clock Rising Edge         | $3T_{osc}$      |                 | ns    |
| $T_{XHGX}$ | Output Data Hold After Clock Rising Edge       | $2T_{osc} - 50$ |                 | ns    |
| $T_{XHGV}$ | Next Output Data Valid After Clock Rising Edge |                 | $2T_{osc} + 50$ | ns    |
| $T_{DVXH}$ | Input Data Setup to Clock Rising Edge          | $3T_{osc} - 30$ |                 | ns    |
| $T_{XHDX}$ | Input Data Hold After Clock Rising Edge        | 0               |                 | ns    |
| $T_{XHGX}$ | Last Clock Rising to Output Float              |                 | $5T_{osc}$      | ns    |

## WAVEFORM—SERIAL PORT—SHIFT REGISTER MODE

## SERIAL PORT WAVEFORM—SHIFT REGISTER MODE

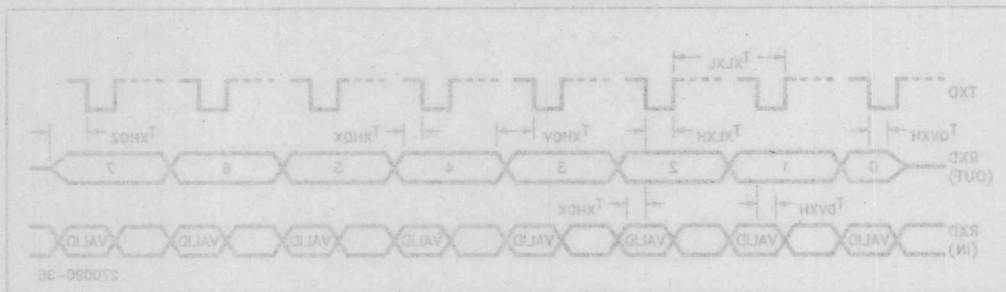


# WAVEFORM — BUSWIDTH PIN



## WAVEFORM—SERIAL PORT—SHIFT REGISTER MODE

SERIAL PORT WAVEFORM—SHIFT REGISTER MODE





## EPROM CHARACTERISTICS

The 879XBH contains 8K bytes of ultraviolet Erasable and Electrically Programmable Read Only Memory (EPROM) for internal storage. This memory can be programmed in a variety of ways — including at run-time under software control.

The EPROM is mapped into memory locations 2000H through 3FFFH if EA is a TTL high. However, applying +12.5V to EA when the chip is reset will place the 879XBH in EPROM Programming mode. The Programming mode has been implemented to support EPROM programming and verification.

When an 879XBH is in Programming mode, special hardware functions are available to the user. These functions include algorithms for slave, gang and auto EPROM programming.

### Programming the 879XBH

Three flexible EPROM programming modes are available on the 879XBH—auto, slave and run-time. These modes can be used to program 879XBHs in a gang, stand alone or run-time environment.

The Auto Programming Mode enables an 879XBH to program itself, and up to 15 other 879XBHs, with the 8K bytes of code beginning at address 4000H on its external bus. The Slave Mode provides a standard interface that enables any number of 879XBHs to be programmed by a master device such as an EPROM programmer. The Run-Time Mode allows individual EPROM locations to be programmed at run-time under complete software control.

In the Programming mode, some I/O pins have been renamed. These new pin functions are used to determine the programming function that is performed, provide programming ALEs, provide slave ID numbers and pass error information. Figure 16 shows

how the pins are renamed. Figure 17 describes each new pin function.

While in Programming mode, PMODE selects the programming function that is performed (see Figure 18). When not in the Programming mode, Run-Time programming can be done at any time.

| PMODE   | Programming Mode           |
|---------|----------------------------|
| 0-4     | Reserved                   |
| 5       | Slave Programming          |
| 6-0BH   | Reserved                   |
| 0CH     | Auto Programming Mode      |
| 0DH     | Program Configuration Byte |
| 0EH-0FH | Reserved                   |

Figure 18. Programming Function PMODE Values

To guarantee proper execution, the pins of PMODE and SID must be in their desired state before the RESET pin is allowed to rise and reset the part. Once the part is reset, it is in the selected mode and should not be switched to another mode without a new reset sequence.

When EA selects the Programming mode, the chip reset sequence loads the CCR from the Programming Chip Configuration Byte (PCCB). This is a separate EPROM location that is not mapped under normal operation. PCCB is only important when programming in the Auto Programming mode. In this mode, the 879XBH that is being programmed gets the data to be programmed from external memory over the system bus. Therefore, PCCR must correctly correspond to the memory system in the programming setup, which is not necessarily the memory organization of the application.

The following sections describe 879XBH programming in each programming mode.

|   |               |
|---|---------------|
| Address/Command/Data Bus. Used to pass commands, addresses and data to and from slave mode 879XBHs. Used by chips in the auto programming mode to pass command, addresses and data to slaves. Also used in the auto programming mode as a regular system bus to access external memory. | PORTS 3 and 4 |
| Slave Programming Pulse. Output from an 879XBH in the auto programming mode. A falling edge on SPPOG indicates that Ports 3 and 4 contain valid data for programming into slave 879XBHs that may be attached to the master.   | SPPOG         |
| Information for slave 879XBHs that may be attached to the master. edge on SALE indicates that Ports 3 and 4 contain valid address/command information for slave 879XBHs that may be attached to the master.   | SALE          |

Figure 17. Programming Mode Pin Definitions

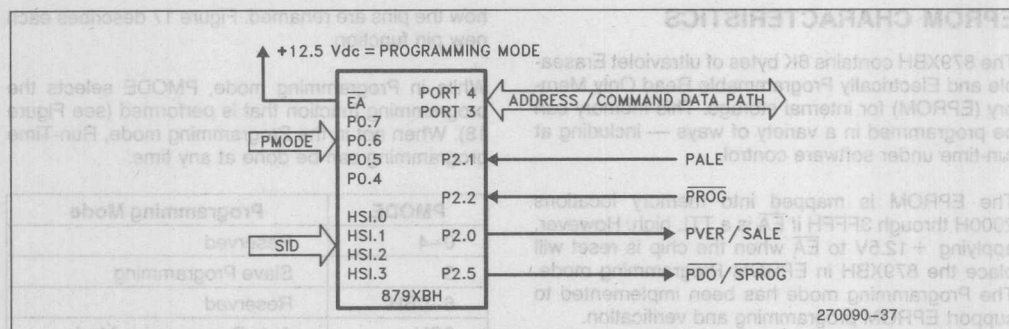


Figure 16. Programming Mode Pin Functions

| Name          | Function  |
|---------------|---|
| PMODE         | Programming Mode Select. Determines the EPROM programming algorithm that is performed. PMODE is sampled after a chip reset and should be static while the part is operating.  |
| SID           | Slave ID Number. Used to assign each slave a pin of Port 3 or 4 to use for passing programming verification acknowledgement. For example, if gang programming in the slave programming mode, the slave with SID = 0001 will use Port 3.1 to signal correct or incorrect program verification.   |
| PALE          | Programming ALE input. Accepted by an 879XBH that is in the slave programming mode. Used to indicate that Ports 3 and 4 contain a command/address.  |
| PROG          | Programming Pulse. Accepted by an 879XBH that is in the slave programming mode. Used to indicate that Ports 3 and 4 contain the data to be programmed. A falling edge on PROG signifies data valid and starts the programming cycle. A rising edge on PROG will halt programming in the slaves. |
| PVER          | Program Verified. A signal output after a programming operation by parts in the slave programming mode.   |
| PDO           | Programming Duration Overflowed. A signal output by parts in the slave programming mode. Used to signify that the PROG pulse applied for a programming operation was longer than allowed.   |
| SALE          | Slave ALE. Output signal from an 879XBH in the auto programming mode. A falling edge on SALE indicates that Ports 3 and 4 contain valid address/command information for slave 879XBHs that may be attached to the master.   |
| SPROG         | Slave Programming Pulse. Output from an 879XBH in the auto programming mode. A falling edge on SPROG indicates that Ports 3 and 4 contain valid data for programming into slave 879XBHs that may be attached to the master.   |
| PORTS 3 and 4 | Address/Command/Data Bus. Used to pass commands, addresses and data to and from slave mode 879XBHs. Used by chips in the auto programming mode to pass command, addresses and data to slaves. Also used in the auto programming mode as a regular system bus to access external memory.         |

Figure 17. Programming Mode Pin Definitions

# AUTO PROGRAMMING MODE

The Auto Programming Mode provides the ability to program the internal 879XBH EPROM without having to use a special EPROM programmer. In this mode, the 879XBH simply programs itself with the data found at external locations 4000H through 5FFFH. All that is required is that some sort of external memory reside at these locations, that  $\overline{EA}$  selects the programming mode and that  $V_{PP}$  is applied. Figure 19 shows a minimum configuration for using an  $8K \times 8$  EPROM to program one 879XBH in the auto programming mode.

The 879XBH first reads a word from external memory, then the Intel intelligent Programming™ Algorithm (described later) is used to program the appropriate EPROM location. Since the erased state of a byte is 0FFH, the Auto Programming Mode will skip locations where the data to be programmed is 0FFH. When all 8k has been programmed, the part outputs a 1 on Port 3.0 if it programmed correctly and a 0 if it failed.

## Gang Programming with the Auto Programming Mode

An 879XBH in the Auto Programming Mode can also be used as a programmer for up to 15 other 879XBHs that are configured in the Slave Programming Mode. To accomplish this, the 879XBH acting

as the master outputs the slave command/data pairs on Ports 3 and 4 necessary to program slave parts with the same data it is programming itself with. Slave ALE (SALE) and Slave PROG (SPROG) signals are provided by the master to the slaves to demultiplex the commands from the data. Figure 20 is a block diagram of a gang programming system using one 879XBH in the Auto Programming Mode. The Slave Programming Mode is described in the next section.

The master 879XBH first reads a word from the external memory controlled by ALE, RD and WR. It then drives Ports 3 and 4 with a Data Program command using the appropriate address and alerts the slaves with a falling edge on SALE. Next, the data to be programmed is driven onto Ports 3 and 4 and slave programming begins with a falling edge on SPROG. At the same time, the master begins to program its own EPROM location with the data read in. Intel's intelligent Programming™ Algorithm is used, with Data Verify commands being given to the slaves after each programming pulse.

When programming is complete, Ports 3 and 4 are driven with all 1s if all parts programmed correctly. Individual bits of Port 3 and 4 will be driven to 0 if the slave with that bit number as an SID did not program correctly. The 879XBH used as the master assigns itself an SID of 0.

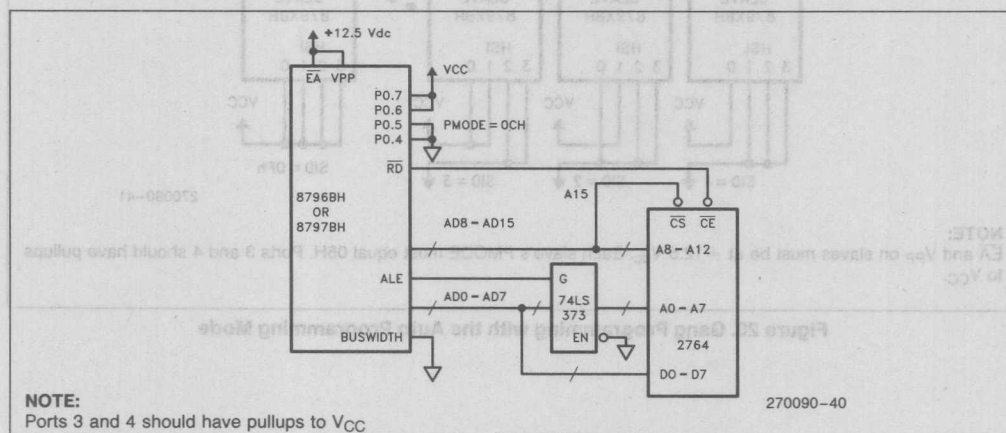
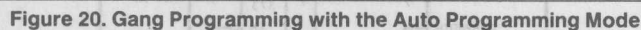


Figure 19. The Auto Programming Mode





## SLAVE PROGRAMMING MODE

Any number of 879XBHs can be programmed by a master programmer through the Slave Programming Mode.

The programming device uses Ports 3 and 4 of the parts being programmed as a command/data path. The slaves accept signals on PALE (Program ALE) and PROG (Program Enable) to demultiplex the commands and data. The slaves also use PVER, PDO and Ports 3 and 4 to pass error information to the programmer. Support for gang programming of up to 16 879XBHs is provided. If each part is given a unique SID (Slave ID Number) an 879XBH in the auto programming mode can be used as a master to program itself and up to 15 other slave 879XBHs. There is, however, no 879XBH dependent limit to the number of parts that can be gang programmed in the slave mode.

## Slave Programming Commands

The commands sent to the slaves are 16-bits wide and contain two fields. Bits 14 and 15 specify the action that the slaves are to perform. Bits 0 through 13 specify the address upon which the action is to take place. Commands are sent via Ports 3 and 4 and are available to cause the slaves to program a word, verify a word, or dump a word (Table 6). The address part of the command sent to the slaves ranges from 2000H to 3FFFH and refers to the internal EPROM memory space. The following sections describe each slave programming mode command.

| P4.7 | P4.6 | Action       |
|------|------|--------------|
| 0    | 0    | Word Dump    |
| 0    | 1    | Data Verify  |
| 1    | 0    | Data Program |
| 1    | 1    | Reserved     |

Table 6. Slave Programming Mode Commands

**DATA PROGRAM COMMAND** — After a Data Program Command has been sent to the slaves, PROG must be pulled low to cause the data on Ports 3 and 4 to be programmed into the location specified during the command. The falling edge of PROG is not only used to indicate data valid, but also triggers the hardware programming of the word specified. The slaves will continue to program the location until PROG rises.

After the rising edge of PROG, the slaves automatically perform a verification of the address just programmed. The result of this verification is then output on PVER (Program Verify) and PDO (Program Duration Overflowed). Therefore, verification information is available following the Data Program Command for programming systems that cannot use the Data Verify command.

If PVER and PDO of all slaves are 1s after PROG rises then the data program was successful everywhere. If PVER is a 0 in any slave, then the data programmed did not verify correctly in that part. If PDO is a 0 in any slave, then the programming pulse in those parts was terminated by an internal safety feature rather than the rising edge of PROG. The safety feature prevents over-programming in the slave mode. Figure 21 shows the relationship of PALE, PROG, PVER and PDO to the Command/Data Path on Ports 3 and 4 for the Data Program Command.

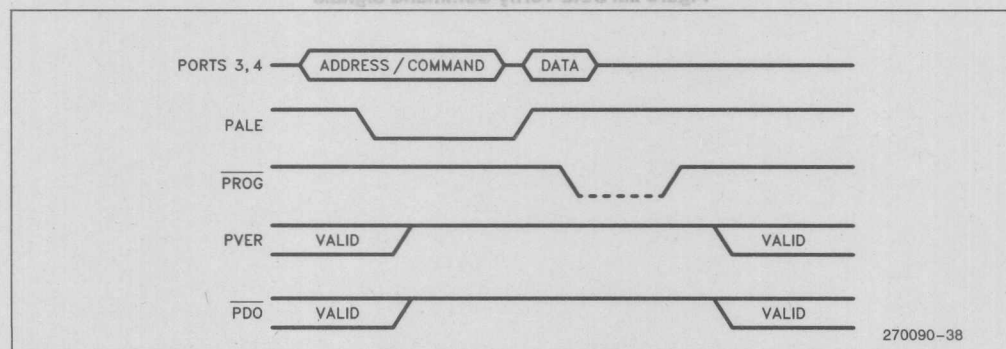


Figure 21. Data Program Signals in Slave Programming Mode

**DATA VERIFY COMMAND** — When the Data Verify Command is sent, the slaves respond by driving one bit of Port 3 or 4 to indicate correct or incorrect verification of the previous Data Program. A 1 indicates correct verification, while a 0 indicates incorrect verification. The SID (Slave ID Number) of each slave determines which bit of the command/data path is driven. PROG from the programmer governs when the slaves drive the bus. Figure 22 shows the relationship of Ports 3 and 4 to PALE and PROG.

This command is usually preceded by a Data Program Command in a programming system with as many as 16 slaves.

**WORD DUMP COMMAND** — When the Word Dump Command is issued, the 879XBH being programmed adds 2000H to the address field of the command and places the value found at the new address on Ports 3 and 4. For example, sending the command #0100H to a slave will result in the slave placing the word found at location 2100H on Ports 3 and 4. PROG from the programmer governs when the slave drives the bus. The signals are the same as shown in Figure 22. This command will work only when just one slave is attached to the bus.

## Gang Programming with the Slave Programming Mode

Gang programming of 879XBHs can be done using the slave programming mode. There is no 879XBH based limit on the number of chips that may be hooked to the same Port 3/Port 4 data path for gang programming.

If more than 16 chips are being gang programmed, the PVER and PDO outputs of each chip could be used for verification. The master programmer could issue a data program command then either watch every chip's error signals, or AND all the signals together to get a system PVER and PDO.

If 16 or fewer 879XBHs are to be gang programmed at once, a more flexible form of verification is available. By giving each chip being programmed a unique SID, the master programmer could then issue a data verify command after the data program command. When a verify command is seen by the slaves, each will drive one pin of Port 3 or 4 with a 1 if the programming verified correctly or a 0 if programming failed. The SID is used by each slave to determine which Port 3, 4 bit it is assigned. An 879XBH in the auto programming mode could be the master programmer if 15 or fewer slaves need to be programmed (See Gang Programming with the Auto Programming Mode).

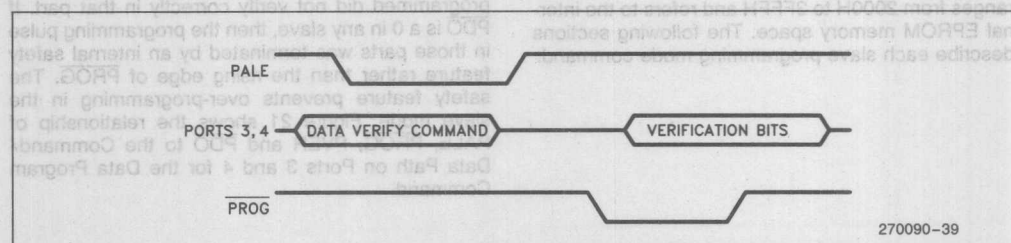


Figure 22. Data Verify Command Signals

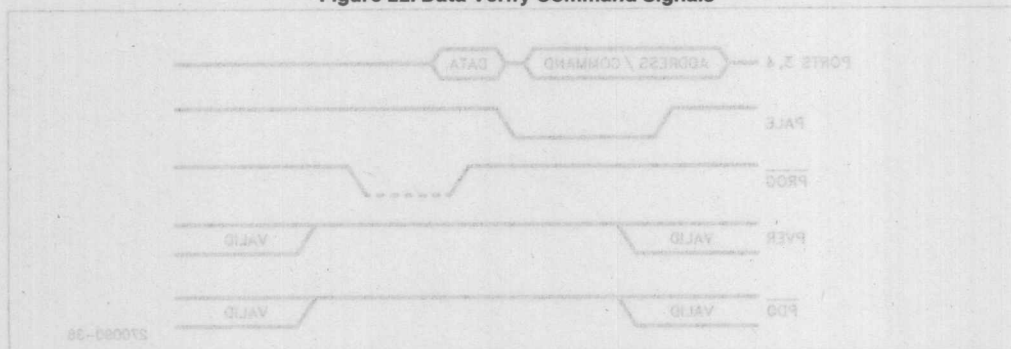


Figure 21. Data Program Signals in Slave Programming Mode

### AUTO CONFIGURATION BYTE PROGRAMMING MODE

The CCB (location 2018H) can be treated just like any other EPROM location, and programmed using any programming mode. But to provide for simple programming of the CCB when no other locations need to be programmed, the Auto Configuration Byte Programming Mode is provided. Programming in this mode also programs PCCB. Figure 23 shows a block diagram for using the Auto Configuration Byte Programming Mode.

With PMODE = 0DH and OFFH on Port 4, CCB and PCCB will be programmed to the value on Port 3 when a logic 0 is placed on PALE. After programming is complete, PVER will be driven to a 1 if the bytes programmed correctly, and a 0 if the programming failed.

This method of programming is the only way to program PCCB. PCCB is a non-memory mapped EPROM location that gets loaded into CCR during the reset sequence when the voltage on EA puts the 879XBH in Programming mode. If PCCB is not programmed using the Auto Configuration Byte Programming Mode, every time the 879XBH is put into Programming mode the CCR will be loaded with 0FFH (the value of the erased PCCB location).

However, if programming of the CCB and PCCB is done using this programming mode, the PCCB will take on the value programmed into CCB. This means that until the part is erased, programming activities that use the system bus will employ the bus width and controls selected by the user's CCB.

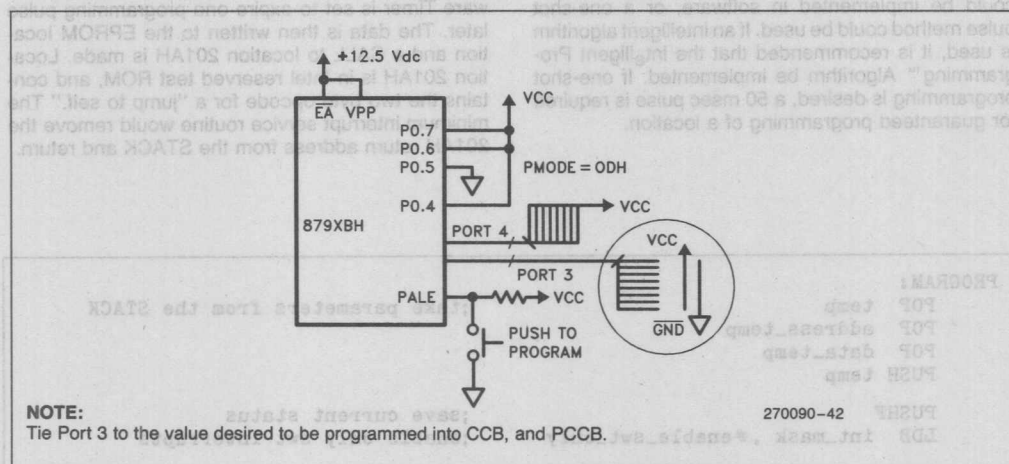


Figure 23. The Auto CCR Programming Mode


**RUN-TIME PROGRAMMING**

Run-Time Programming of the 879XBH is provided to allow the user complete flexibility in the ways in which the internal EPROM is programmed. That flexibility includes the ability to program just one byte or one word instead of the whole EPROM, and extends to the hardware necessary to program. The only additional requirement of a system is that a programming voltage is applied to Vpp. Run-Time Programming is done with EA at TTL-high (normal operation — internal/external access). To Run-Time program, the user writes a byte or word to the location to be programmed. Once this is done, the 879XBH will continue to program that location until another data read from or data write to the EPROM occurs. The user can therefore control the duration of the programming pulse to within a few microseconds. An intelligent programming algorithm could be implemented in software, or a one-shot pulse method could be used. If an intelligent algorithm is used, it is recommended that the intelligent Programming™ Algorithm be implemented. If one-shot programming is desired, a 50 msec pulse is required for guaranteed programming of a location.

After the programming of a location has started, care must be taken to insure that no program fetches (or pre-fetches) occur from internal memory. This is of no concern if the program is executing from external memory. However, if the program is executing from internal memory when the write occurs, it will be necessary to use the built in "jump to self" located at 201AH.

"Jump to Self" is a two byte instruction in the Intel test ROM which can be CALLED after the user has started programming a location by writing to it. A software timer interrupt could then be used to escape from the "jump to self" when the proper programming pulse duration has elapsed. Figure 24 is an example of how to program an EPROM location while execution is entirely internal.

Upon entering the PROGRAM routine, the address and data are retrieved from the STACK and a Software Timer is set to expire one programming pulse later. The data is then written to the EPROM location and a CALL to location 201AH is made. Location 201AH is in Intel reserved test ROM, and contains the two byte opcode for a "jump to self." The minimum interrupt service routine would remove the 201AH return address from the STACK and return.

**PROGRAM:**


```

POP temp
POP address_temp
POP data_temp
PUSH temp

PUSHF
LDB int_mask, #enable_swt_only

LDB HSO_COMMAND, #SWT0_ovf
ADD HSO_TIME, TIMER1, #program_pulse

EI
ST data_temp, [address_temp]
CALL 201AH

POPF
RET

SWT_ISR:
...

swt0_expired:
POP 0
RET
...
```

;take parameters from the STACK

;save current status

;enable only swt interrupts

;load swt command to interrupt

;when program pulse time

;has elapsed

Figure 24. Programming the EPROM from Internal Memory Execution



## The intelligent Programming™ Algorithm

The 879XBH intelligent Programming™ Algorithm rapidly programs the EPROMs of Intel 879XBHs using an efficient and reliable method particularly suited to the production programming environment. Typical programming times for individual devices are on the order of 30 seconds. Programming reliability is also ensured as the incremental program margin of each byte is continually monitored to determine when it has been successfully programmed.

The intelligent Programming™ Algorithm utilizes two different pulse types: initial and overprogram. Repeated initial (250 state time) pulses are applied until the location being programmed verifies correct, where the number of pulses necessary is X. Then, one overprogram pulse is applied with a duration that is 3X as long as one initial pulse. Up to 25 initial pulses per location are provided before the overprogram pulse is applied. If more than 25 pulses are required, programming has failed.

*The entire sequence of program pulses and verification is performed at  $V_{CC} = 6.0V$  and  $V_{PP} = EA = 12.5V$ .*

When the intelligent Programming™ cycle has been completed, all locations should be compared to the original data with  $V_{CC} = V_{PP} = EA = 5.0V$ .

## Erasing the 879XBH EPROM

Initially, and after each erasure, all bits of the 879XBH are in the "1" state. Data is introduced by

selectively programming "0s" into the desired bit locations. Although only "0s" will be programmed, both "1s" and "0s" can be present in the data word. The only way to change a "0" to a "1" is by ultraviolet light erasure.

The erasure characteristics of the 879XBH are such that erasure begins to occur upon exposure to light with wavelengths shorter than approximately 4000 Angstroms (Å). It should be noted that sunlight and certain types of fluorescent lamps have wavelengths in the 3000–4000 Å range. Constant exposure to room level fluorescent lighting could erase the typical 879XBH in approximately 3 years, while it would take approximately 1 week to cause erasure when exposed to direct sunlight. If the 879XBH is to be exposed to these types of lighting conditions for extended periods of time, opaque labels should be placed over the EPROM's window to prevent unintentional erasure.

The recommended erasure procedure for the 879XBH is exposure to shortwave ultraviolet light which has a wavelength of 2537 Å. The integrated dose (i.e., UV intensity  $\times$  exposure time) for erasure should be a minimum of 15 Wsec/cm<sup>2</sup>. The erasure time with this dosage is approximately 15 to 20 minutes using an ultraviolet lamp with a 12000  $\mu W/cm^2$  power rating. The 879XBH should be placed within 1 inch of the lamp tubes during erasure. The maximum integrated dose an 879XBH can be exposed to without damage is 7258 Wsec/cm<sup>2</sup> (1 week @ 12000  $\mu W/cm^2$ ). Exposure of the 879XBH to high intensity UV light for long periods may cause permanent damage.

| Symbol   | Parameter                                      |
|----------|--|
| $I_{PP}$ | $V_{PP}$ Supply Current (Whenever Programming) |
| 50       | mA   |

NOTE:  $V_{CC}$  must be at least 4.5V before  $V_{PP}$  rises above 5V.  $V_{PP}$  must not have a low impedance path to VEE while  $V_{CC} > 4.5V$ . The A/D converter accuracy is not guaranteed when  $V_{PP}$  is above 8.5V.

## EPROM SPECIFICATIONS

### A.C. EPROM PROGRAMMING CHARACTERISTICS

Test Conditions: Load Capacitance = 150 pF,  $T_a = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ,  $V_{CC} = V_{PD} = V_{REF} = 6.0\text{V} \pm .5\text{V}$ ,  $V_{PP} = 12.5\text{V} \pm .5\text{V}$ ,  $\overline{EA} = 11\text{V} \pm 2.0\text{V}$ ,  $f_{osc} = 6.0$  to  $12.0$  MHz

| Symbol     | Parameter  | Min           | Max    | Units     |
|------------|--|---------------|--------|-----------|
| $T_{AVLL}$ | ADDRESS/COMMAND Valid to PALE Low                      | 0             |        | $T_{osc}$ |
| $T_{LLAX}$ | ADDRESS/COMMAND Hold After PALE Low                    | 80            |        | $T_{osc}$ |
| $T_{DVPL}$ | Output Data Setup Before $\overline{PROG}$ Low         | 0             |        | $T_{osc}$ |
| $T_{PLDX}$ | Data Hold After $\overline{PROG}$ Falling              | 80            |        | $T_{osc}$ |
| $T_{LLLH}$ | PALE Pulse Width                                       | 180           |        | $T_{osc}$ |
| $T_{PLPH}$ | $\overline{PROG}$ Pulse Width                          | $250 T_{osc}$ | 100 ms |           |
| $T_{LHPL}$ | PALE High to $\overline{PROG}$ Low                     | 250           |        | $T_{osc}$ |
| $T_{PHLL}$ | $\overline{PROG}$ High to Next PALE Low                | 600           |        | $T_{osc}$ |
| $T_{PHDX}$ | Data Hold After $\overline{PROG}$ High                 | 100           |        | $T_{osc}$ |
| $T_{PHVV}$ | $\overline{PROG}$ High to PVER/ $\overline{PD0}$ Valid | 500           |        | $T_{osc}$ |
| $T_{LLVH}$ | PALE Low to PVER/ $\overline{PD0}$ High                | 100           |        | $T_{osc}$ |
| $T_{PLDV}$ | $\overline{PROG}$ Low to VERIFICATION/DUMP Data Valid  | 100           |        | $T_{osc}$ |
| $T_{SHLL}$ | RESET High to First PALE Low (not shown)               | 2000          |        | $T_{osc}$ |

#### NOTE:

Run-time programming is done with  $V_{CC} = V_{PD} = V_{REF} = 5\text{V}$  and  $V_{PP} = 12.5\text{V}$ . One 50 msec programming pulse in run-time mode is guaranteed to program a location.

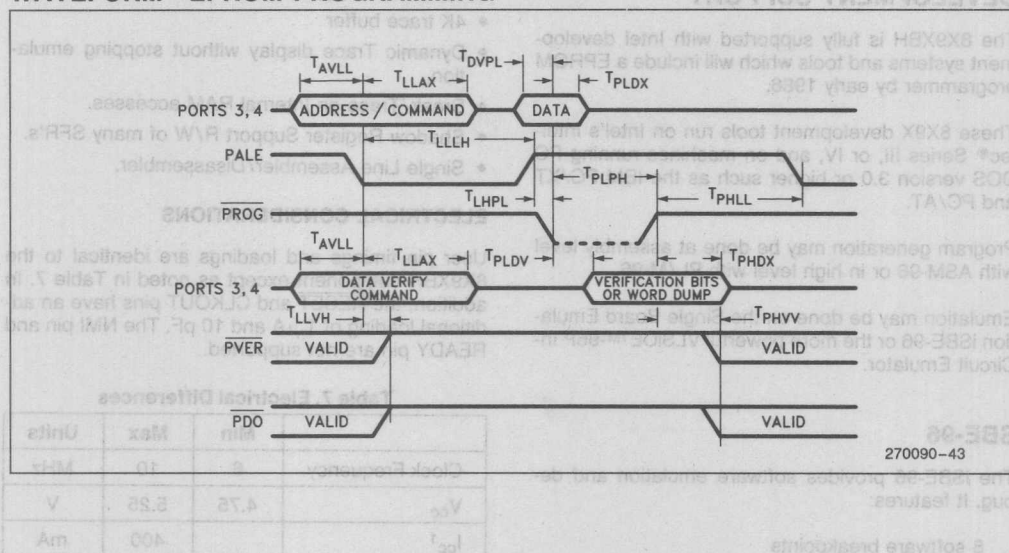
### D.C. EPROM PROGRAMMING CHARACTERISTICS

| Symbol   | Parameter                                      | Min | Max | Units |
|----------|--|-----|-----|-------|
| $I_{PP}$ | $V_{PP}$ Supply Current (Whenever Programming) |     | 50  | mA    |

#### NOTE:

$V_{CC}$  must be at least 4.5V before  $V_{PP}$  rises above 5V.  $V_{PP}$  must not have a low impedance path to  $V_{SS}$  while  $V_{CC} > 4.5\text{V}$ . The A/D Converter accuracy is not guaranteed when  $V_{REF}$  is above 5.5V.

## WAVEFORM—EPROM PROGRAMMING



### MECHANICAL CONSIDERATIONS

VLSICE™-88P comes complete with target adapter which plug directly into standard 48 pin DIP and 68 pin PGA sockets.

The user plug is at the end of a two foot flexible cable. Adequate spacing must be provided on the target system to allow the emulation processor board and user plug to be inserted into the target system.

Figure 25 shows the physical dimensions and orientation of pin 1 for both the 68 pin PGA and 48 pin DIP versions of the user plug.

### VLSICE™-88P IN-CIRCUIT EMULATOR

The more powerful VLSICE-88 is an in-circuit emulator for with Intel's bond-out technology.

- Precise real-time emulation
- 64K of mappable I/O memory

## DEVELOPMENT SUPPORT

The 8X9XBH is fully supported with Intel development systems and tools which will include a EPROM programmer by early 1986.

These 8X9X development tools run on Intel's Inteltec® Series III, or IV, and on machines running PC DOS version 3.0 or higher such as the IBM PC/XT and PC/AT.

Program generation may be done at assembly level with ASM-96 or in high level with PL/M-96.

Emulation may be done on the Single Board Emulation iSBE-96 or the more powerful VLSiCETM-96P In-Circuit Emulator.

## SBE-96

The iSBE-96 provides software emulation and debug. It features:

- 8 software breakpoints
- 12 MHz emulation speed
- Configurable serial I/O
- Single Line Assembly/Disassembly

It supports all three package types (48 pin dip, 68 pin PGA, 68 pin PLCC via two 50 pin ribbon cables and adaptor boards).

## VLSiCETM-96P IN-CIRCUIT EMULATOR

The more powerful VLSiCE-96 is an in-circuit emulator with Intel's bond-out featuring:

- Precise realtime emulation
- 64K of mappable ICE memory

- Symbolic Debugging
- 4K trace buffer
- Dynamic Trace display without stopping emulation.
- Break/Trace on Internal RAM accesses.
- Shadow Register Support R/W of many SFR's.
- Single Line Assembler/Disassembler.

## ELECTRICAL CONSIDERATIONS

User pin timings and loadings are identical to the 8X9XBH component except as noted in Table 7. In addition, the RESET and CLKOUT pins have an additional loading of 1  $\mu$ A and 10 pF. The NMI pin and READY pin are not supported.

Table 7. Electrical Differences

|                              | Min  | Max  | Units |
|------------------------------|------|------|-------|
| Clock Frequency              | 6    | 10   | MHz   |
| V <sub>cc</sub>              | 4.75 | 5.25 | V     |
| I <sub>cc</sub> <sup>1</sup> |      | 400  | mA    |

<sup>1</sup>All outputs disconnected

## MECHANICAL CONSIDERATIONS

VLSiCETM-96P comes complete with target adaptors which plug directly into standard 48 pin DIP and 68 pin PGA sockets.

The user plug is at the end of a two foot flexible cable. Adequate spacing must be provided on the target system to allow the emulation processor board and user plug to be inserted into the target system.

Figure 25 shows the physical dimensions and orientation of pin 1 for both the 68 pin PGA and 48 pin DIP versions of the user plug.



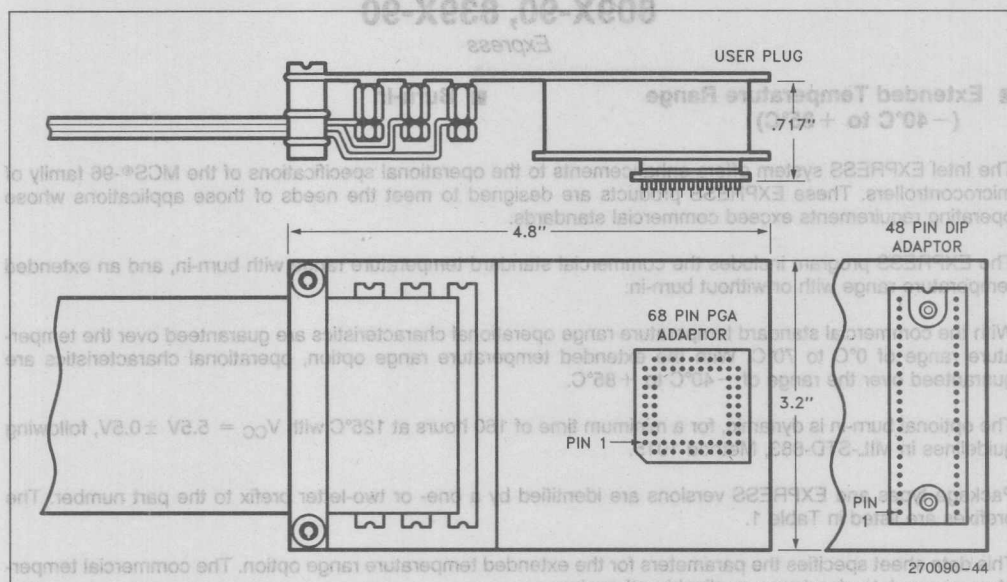


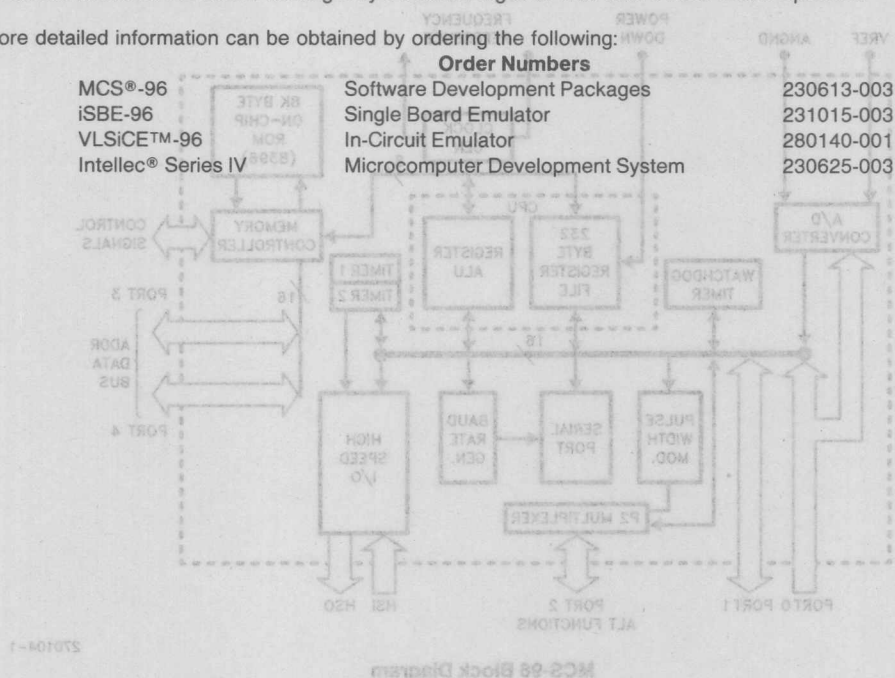
Figure 25. VLSiC™-96P User Plug Dimensions

All combinations of the above hosting may be linked together with Intel's NDS II and OpenNet.

More detailed information can be obtained by ordering the following:

**Order Numbers**

|                     |                                  |            |
|---------------------|----------------------------------|------------|
| MCS®-96             | Software Development Packages    | 230613-003 |
| iSBE-96             | Single Board Emulator            | 231015-003 |
| VLSiC™-96           | In-Circuit Emulator              | 280140-001 |
| Intellec® Series IV | Microcomputer Development System | 230625-003 |



# 809X-90, 839X-90

Express

## Extended Temperature Range (-40°C to +85°C)

## Burn-In

The Intel EXPRESS system offers enhancements to the operational specifications of the MCS®-96 family of microcontrollers. These EXPRESS products are designed to meet the needs of those applications whose operating requirements exceed commercial standards.

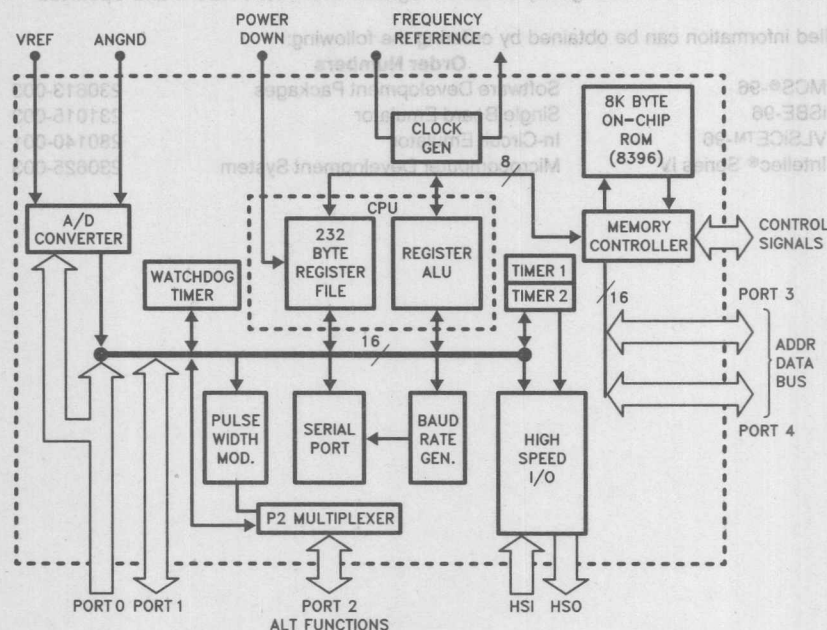
The EXPRESS program includes the commercial standard temperature range with burn-in, and an extended temperature range with or without burn-in.

With the commercial standard temperature range operational characteristics are guaranteed over the temperature range of 0°C to 70°C. With the extended temperature range option, operational characteristics are guaranteed over the range of -40°C to +85°C.

The optional burn-in is dynamic, for a minimum time of 160 hours at 125°C with  $V_{CC} = 5.5V \pm 0.5V$ , following guidelines in MIL-STD-883, Method 1015.

Package types and EXPRESS versions are identified by a one- or two-letter prefix to the part number. The prefixes are listed in Table 1.

This data sheet specifies the parameters for the extended temperature range option. The commercial temperature range data sheets are applicable otherwise.



MCS-96 Block Diagram

270104-1

## ELECTRICAL CHARACTERISTICS ABSOLUTE MAXIMUM RATINGS\*

Ambient Temperature Under Bias . . . -40°C to +85°C  
Storage Temperature . . . . . -40°C to +150°C  
Voltage from Any Pin to  
VSS or ANGND . . . . . -0.3V to +7.0V  
Average Output Current from Any Pin . . . . . 10 mA  
Power Dissipation . . . . . 1.5W

\*Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

NOTICE: Specifications contained within the following tables are subject to change.

## OPERATING CONDITIONS

| Symbol           | Parameter                      | Min | Max | Units |
|------------------|--------------------------------|-----|-----|-------|
| T <sub>A</sub>   | Ambient Temperature Under Bias | -40 | +85 | °C    |
| V <sub>CC</sub>  | Digital Supply Voltage         | 4.5 | 5.5 | V     |
| V <sub>REF</sub> | Analog Supply Voltage          | 4.5 | 5.5 | V     |
| f <sub>OSC</sub> | Oscillator Frequency           | 6.0 | 12  | MHz   |
| V <sub>PD</sub>  | Power-Down Supply Voltage      | 4.5 | 5.5 | V     |

### NOTE:

V<sub>BB</sub> should be connected to ANGND through a 0.01 µF capacitor. ANGND and V<sub>SS</sub> should be nominally at the same potential.

## D.C. CHARACTERISTICS T<sub>A</sub> = -40°C to +85°C

| Symbol           | Parameter   | Min  | Max                   | Units | Test Conditions                        |
|------------------|---|------|-----------------------|-------|--|
| V <sub>IL</sub>  | Input Low Voltage (Except RESET)                                  | -0.3 | +0.8                  | V     |  |
| V <sub>IL1</sub> | Input Low Voltage, RESET  | -0.3 | +0.7                  | V     |  |
| V <sub>IH</sub>  | Input High Voltage (Except RESET, NMI, XTAL1)                     | 2.0  | V <sub>CC</sub> + 0.5 | V     |  |
| V <sub>IH1</sub> | Input High Voltage, NMI, XTAL1, RESET                             | 2.4  | V <sub>CC</sub> + 0.5 | V     |  |
| V <sub>OL</sub>  | Output Low Voltage  |      | 0.5                   | V     | (Note 1)                               |
| V <sub>OH</sub>  | Output High Voltage   | 2.4  |                       | V     | (Note 2)                               |
| I <sub>CC</sub>  | V <sub>CC</sub> Supply Current                                    |      | 200                   | mA    | All Outputs Disconnected               |
| I <sub>PD</sub>  | V <sub>PD</sub> Supply Current                                    |      | 1                     | mA    | Normal Operation and Power-Down        |
| I <sub>REF</sub> | V <sub>REF</sub> Supply Current                                   |      | 10                    | mA    |  |
| I <sub>LB</sub>  | Input Leakage Current to All Pins of HSI, P0, P3, P4, and to P2.1 |      | ±10                   | µA    | V <sub>in</sub> = 0 to V <sub>CC</sub> |
| I <sub>IH</sub>  | Input High Current to $\overline{EA}$                             |      | 100                   | µA    | V <sub>IH</sub> = 2.4V                 |
| I <sub>IL</sub>  | Input Low Current to All Pins of P1, and to P2.6, P2.7            |      | -100                  | µA    | V <sub>IL</sub> = 0.45V                |
| I <sub>IL1</sub> | Input Low Current to RESET  |      | -2                    | mA    | V <sub>IL</sub> = 0.45V                |
| I <sub>IL2</sub> | Input Low Current P2.2, P2.3, P2.4, READY                         |      | -50                   | µA    | V <sub>IL</sub> = 0.45V                |
| C <sub>s</sub>   | Pin Capacitance (Any Pin to V <sub>SS</sub> )                     |      | 10                    | pF    | f <sub>TEST</sub> = 1 MHz              |

### NOTES:

1. I<sub>OL</sub> = 0.4 mA for all pins of P1, for P2.6 and P2.7, and for all pins of P3 and P4 when used as ports. I<sub>OL</sub> = 2.0 mA for TXD, RSD (in serial port mode 0), PWM, CLKOUT, ALE, BHE, RD, WR, and all pins of HSO and P3 and P4 when used as external memory bus (AD0-AD15).
2. I<sub>OH</sub> = -20 µA for all pins of P1, for P2.6 and P2.7. I<sub>OH</sub> = -200 µA for TXD, RXD (in serial port mode 0), PWM, CLKOUT, ALE, BHE, WR, and all pins of HSO and P3 and P4 when used as external memory bus (AD0-AD15). P3 and P4, when used as ports, have open-drain outputs.

# A/D CONVERTER SPECIFICATIONS

A/D Converter operation is verified only on the 8097, 8397, 8095, 8395.

The absolute conversion accuracy is dependent on the accuracy of  $V_{REF}$ . The specifications given below assume adherence to the Operating Conditions section of these data sheets. Testing is done at  $V_{REF} = 5.120V$ .

|                           |                                 |
|---------------------------|---------------------------------|
| Resolution                | $\pm 0.001 V_{REF}$             |
| Accuracy                  | $\pm 0.004 V_{REF}$             |
| Differential nonlinearity | $\pm 0.002 V_{REF} \text{ max}$ |
| Integral nonlinearity     | $\pm 0.004 V_{REF} \text{ max}$ |

# A.C. CHARACTERISTICS $V_{CC}, V_{PD} = 4.5V \text{ to } 5.5V, T_A = -40^\circ C \text{ to } +85^\circ C; f_{osc} = 6.0 \text{ MHz to } 12.0 \text{ MHz}$

Test Conditions: Load capacitance on output pins = 80 pF

Oscillator Frequency = 12.00 MHz

# TIMING REQUIREMENTS Other system components must meet these specs

| Symbol | Parameter   | Min        | Max                 | Units |
|--------|---|------------|---------------------|-------|
| TCLYX  | READY Hold after CLKOUT Falling Edge                  | 0 (Note 1) |                     | ns    |
| TLLYV  | End of ALE to READY Setup                             | -Tosc      | 2Tosc - 60          | ns    |
| TLLYH  | End of ALE to READY High                              | 2Tosc + 60 | 4Tosc - 60 (Note 2) | ns    |
| TYLYH  | Non-Ready Time  |            | 1000                | ns    |
| TAVDV  | Address Valid to Input Data Valid                     |            | 5Tosc - 90          | ns    |
| TRLDV  | $\overline{RD}$ Active to Input Data Valid            |            | 3Tosc - 60          | ns    |
| TRXDX  | Data Hold after $\overline{RD}$ Inactive (Note 3)     | 0          |                     | ns    |
| TRXDZ  | $\overline{RD}$ Inactive to Input Data Float (Note 3) |            | Tosc - 20           | ns    |

# TIMING RESPONSES MCS-96 parts meet these specs

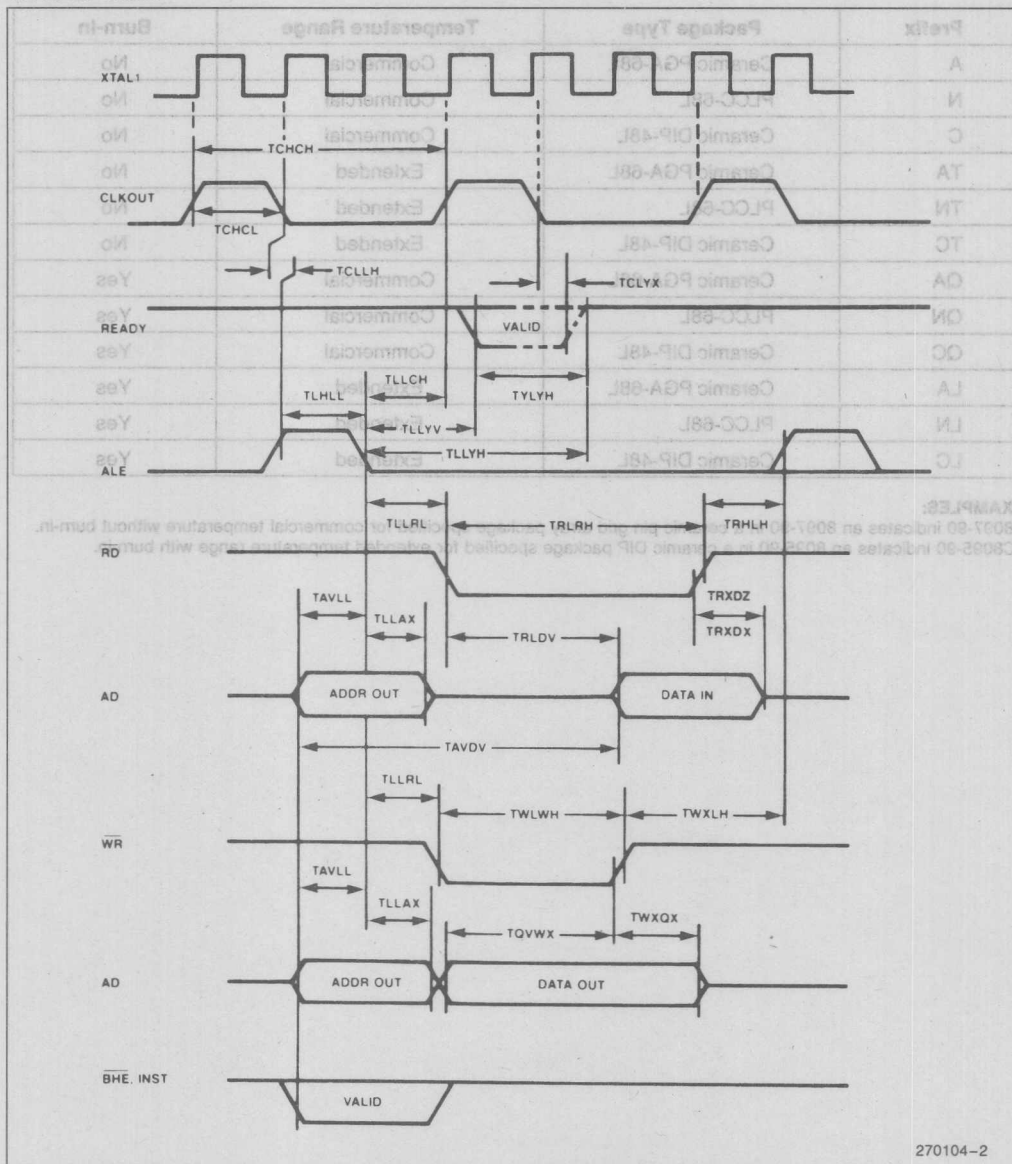
| Symbol | Parameter   | Min            | Max            | Units |
|--------|---|----------------|----------------|-------|
| FXTAL  | Oscillator Frequency                                    | 6.00           | 12.00          | MHz   |
| Tosc   | Oscillator Period                                       | 83             | 166            | ns    |
| TCHCH  | CLKOUT Period (Note 3)                                  | 3Tosc (Note 4) | 3Tosc (Note 4) | ns    |
| TCHCL  | CLKOUT High Time  | Tosc - 20      | Tosc + 20      | ns    |
| TCLLH  | CLKOUT Low to ALE High                                  | -10            | 30             | ns    |
| TLLCH  | ALE Low to CLKOUT High                                  | Tosc - 20      | Tosc + 40      | ns    |
| TLHLL  | ALE Pulse Width   | Tosc - 25      | Tosc + 20      | ns    |
| TAVLL  | Address Setup to End of ALE                             | Tosc - 50      |                | ns    |
| TLLRL  | End of ALE to $\overline{RD}$ or $\overline{WR}$ Active | Tosc - 20      |                | ns    |
| TLLAX  | Address Hold after End of ALE                           | Tosc - 20      |                | ns    |
| TWLWH  | $\overline{WR}$ Pulse Width                             | 2Tosc - 35     |                | ns    |
| TQVWX  | Output Data Setup to End of $\overline{WR}$             | 2Tosc - 60     |                | ns    |
| TWXQX  | Output Data Hold after End of $\overline{WR}$           | Tosc - 25      |                | ns    |
| TWXLH  | End of $\overline{WR}$ to Next ALE                      | 2Tosc - 30     |                | ns    |
| TRLRH  | $\overline{RD}$ Pulse Width                             | 3Tosc - 30     |                | ns    |
| TRHLH  | End of $\overline{RD}$ to Next ALE                      | Tosc - 30      |                | ns    |

## NOTES:

1. If the 48-pin part is being used then this timing can be generated by assuming that the CLKOUT falling edge has occurred at  $2Tosc + 60$  (TLLCH(max) + TCHCL(max)) after the falling edge of ALE.
2. If more than one wait state is desired, add 3Tosc for each additional wait state.
3. This specification is not tested, but is verified by design analysis and/or derived from other tested parameters.
4. CLKOUT is directly generated as a divide by 3 of the oscillator. The period will be  $3Tosc \pm 10$  ns if Tosc is constant and the rise and fall times on XTAL 1 are less than 10 ns.



WAVEFORM



Bus Signal Timings

270104-2

**EXAMPLES:**

---

**MCS<sup>®</sup>-96**  
**Application Note**

---

**6**

6

MCS®-9d  
Application Note

---



High speed digital signals are frequently encountered in modern control applications. In addition, there is often a requirement for high speed 16-bit and 32-bit precision in calculations. The MCS-86 product line, generally referred to as the 8096, is designed to be used in applications which require high speed calculations and fast I/O operations.

The 8096 is a 16-bit microcontroller with dedicated I/O subsystems and a complete set of 16-bit arithmetic including multiply and divide operations. This paper will briefly describe the 8096 in section 2, and then give short examples of how to use each of its key features in section 3. The concluding sections feature a few examples which make use of several chip features simultaneously and some hardware connection suggestions. Further information on the 8096 and its use is available from the sources listed in the bibliography.

February 1985

## 2.1 GENERAL DESCRIPTION

Unlike microprocessors, microcontrollers are generally optimized for specific applications. Intel's 8048 was optimized for general control tasks while the 8051 was optimized for 8-bit math and single bit boolean operations. The 8096 has been designed for high speed/high performance control applications. Because it has been designed for these applications the 8096 architecture is different from that of the 8048 or 8051.

There are two major sections of the 8096: the CPU section and the I/O section. Each of these sections can be subdivided into functional blocks as shown in Figure 2-1.

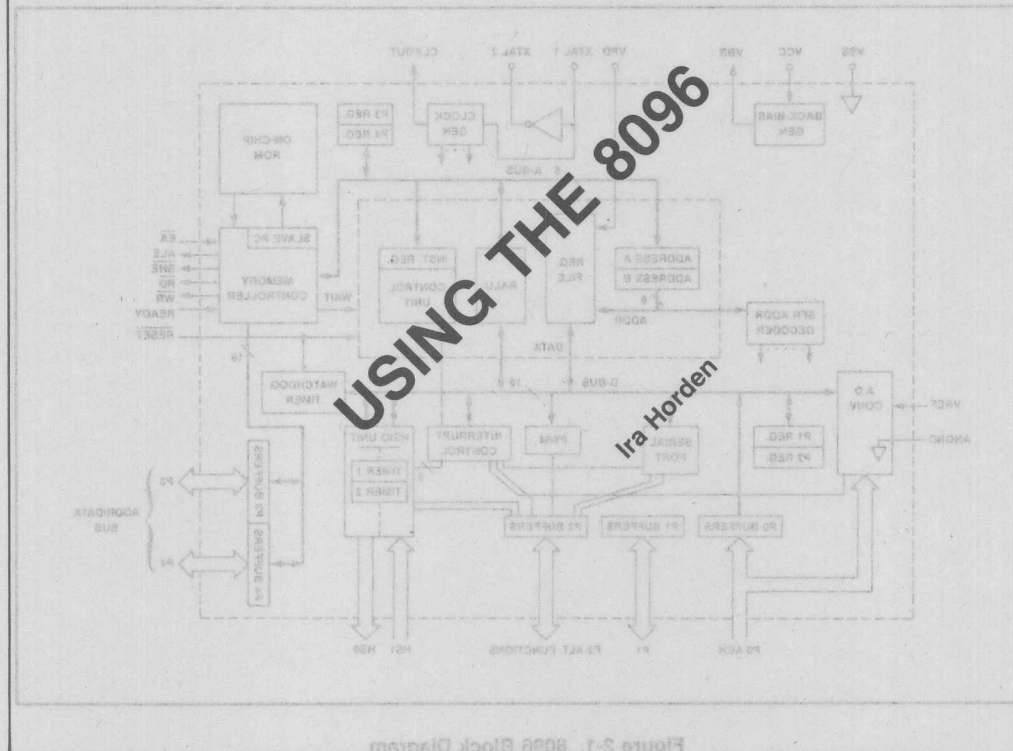


Figure 2-1. 8096 Block Diagram

## 1.0 INTRODUCTION

High speed digital signals are frequently encountered in modern control applications. In addition, there is often a requirement for high speed 16-bit and 32-bit precision in calculations. The MCS-96 product line, generically referred to as the 8096, is designed to be used in applications which require high speed calculations and fast I/O operations.

The 8096 is a 16-bit microcontroller with dedicated I/O subsystems and a complete set of 16-bit arithmetic instructions including multiply and divide operations. This Ap-note will briefly describe the 8096 in section 2, and then give short examples of how to use each of its key features in section 3. The concluding sections feature a few examples which make use of several chip features simultaneously and some hardware connection suggestions. Further information on the 8096 and its use is available from the sources listed in the bibliography.

## 2.0 8096 OVERVIEW

### 2.1. GENERAL DESCRIPTION

Unlike microprocessors, microcontrollers are generally optimized for specific applications. Intel's 8048 was optimized for general control tasks while the 8051 was optimized for 8-bit math and single bit boolean operations. The 8096 has been designed for high speed/high performance control applications. Because it has been designed for these applications the 8096 architecture is different from that of the 8048 or 8051.

There are two major sections of the 8096: the CPU section and the I/O section. Each of these sections can be subdivided into functional blocks as shown in Figure 2-1.

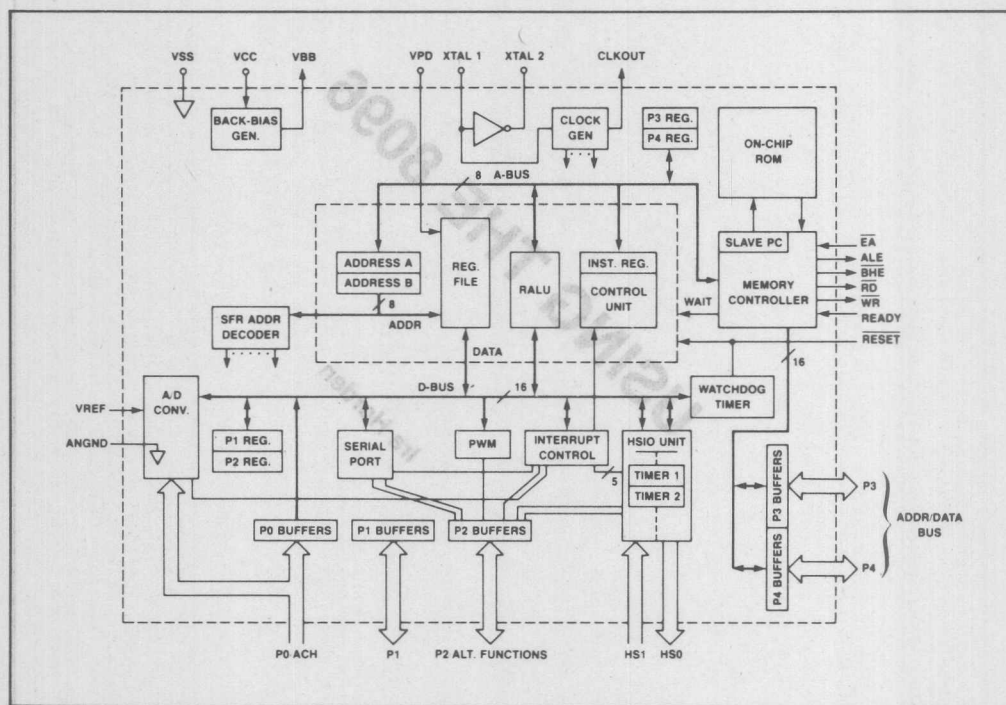
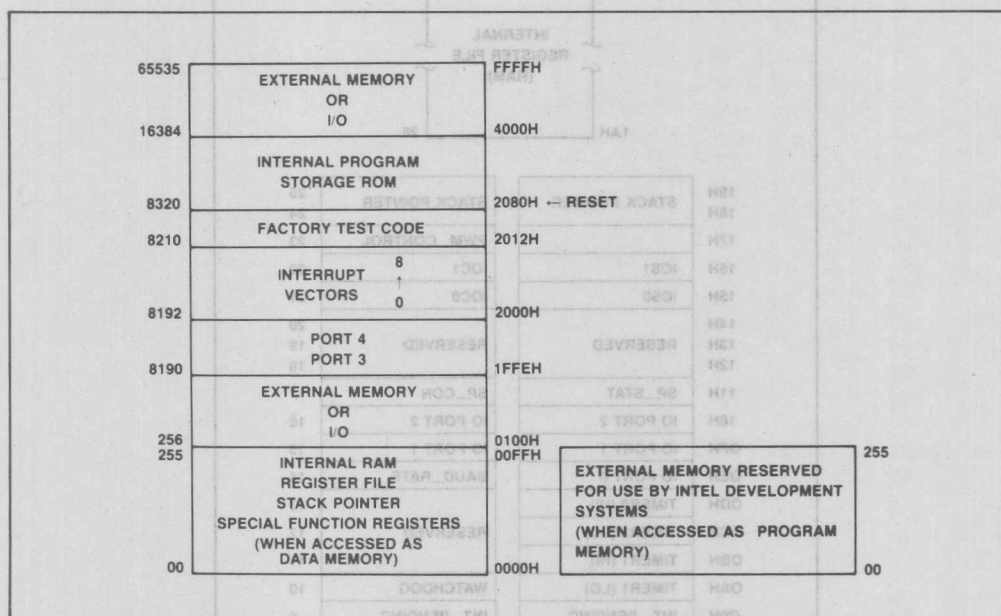


Figure 2-1. 8096 Block Diagram

### 2.1.1. CPU section

The CPU of the 8096 uses a 16-bit ALU which operates on a 256-byte register file instead of an accumulator. Any of the locations in the register file can be used for sources or destinations for most of the instructions. This is called a register to register architecture. Many of the instructions can also use bytes or words from anywhere in the 64K byte address space as operands. A memory map is shown in Figure 2-2.



### Figure 2-2. Memory Map

Figure 2-3 shows the layout of the register mapped I/O. Some of these registers serve two functions, one if they are read from and another if they are written

to. More information about the use of these registers is included in the description of the features which they control.

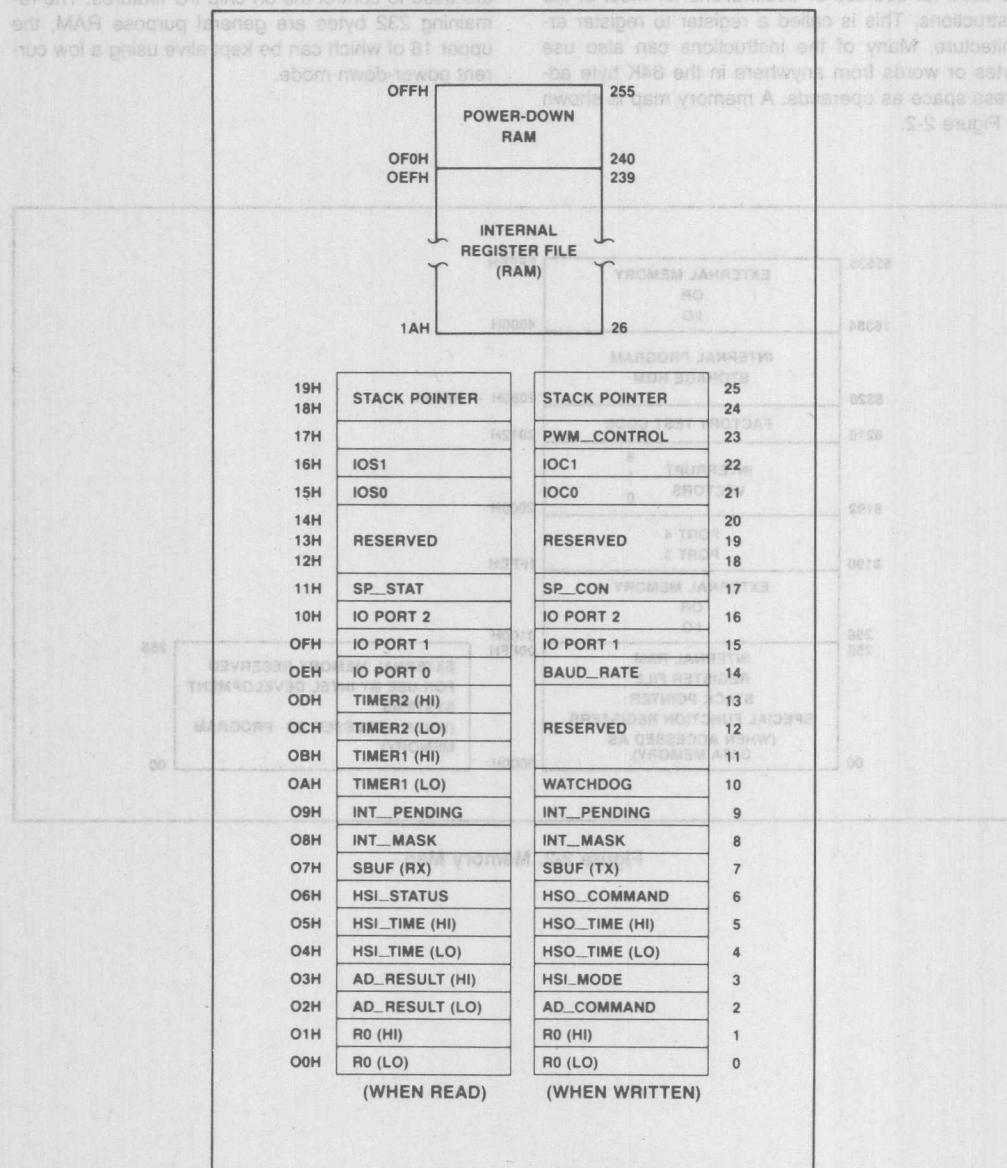


Figure 2-3: SFR Layout



## 2.1.2. I/O Features

Many of the I/O features on the 8096 are designed to operate with little CPU intervention. A list of the major I/O functions is shown in Figure 2-4. The Watchdog Timer is an internal timer which can be used to reset the system if the software fails to operate properly. The Pulse-Width-Modulation (PWM) output can be used as a rough D to A, a motor driver, or for many other purposes. The A to D converter (ADC) has 8 multiplexed inputs and 10-bit resolution. The serial port has several modes and its own baud rate generator. The High Speed I/O section includes a 16-bit timer, a 16-bit counter, a 4-input programmable edge detector, 4 software timers, and a 6-output programmable event generator. All of these features will be described in section 2.3.

## 2.2. THE PROCESSOR SECTION

### 2.2.1. Operations and addressing modes

The 8096 has 100 instructions, some of which operate on bits, some on bytes, some on words and some on longs (double words). All of the standard logical and arithmetic functions are available for both byte and word operations. Bit operations and long operations are provided for some instructions. There are also flag manipulation instructions as well as jump and call instructions. A full set of conditional jumps has been included to speed up testing for various conditions.

Bit operations are provided by the Jump Bit and Jump Not Bit instructions, as well as by immediate masking of bytes. These bit operations can be performed on any of the bytes in the register file or on any of the special function registers. The fast bit manipulation of the SFRs can provide rapid I/O operations.

A symmetric set of byte and word operations make up the majority of the 8096 instruction set. The assembly language for the 8096 (ASM-96) uses a "B" suffix on a mnemonic to indicate a byte operation, without this suffix a word operation is indicated. Many of these operations can have one two or three operands. An example of a one operand instruction would be:

NOT Value1 ; Value1 := 1's complement (Value1)

A two operand instruction would have the form:

ADD Value2,Value1 ; Value2 := Value2 + Value1

A three operand instruction might look like:

MUL Value3,Value2,Value1 ;  
Value3 := Value2 \* Value1

The three operand instructions combined with the register to register architecture almost eliminate the necessity of using temporary registers. This results in a faster processing time than machines that have equivalent instruction execution times, but use a standard architecture.

Long (32-bit) operations include shifts, normalize, and multiply and divide. The word divide is a 32-bit by 16-bit operation with a 16-bit quotient and 16-bit remainder. The word multiply is a word by word multiply with a long result. Both of these operations can be done in either the signed or unsigned mode. The direct unsigned modes of these instructions take only 6.5 microseconds. A normalize instruction and sticky bit flag have been included in the instruction set to provide hardware support for the software floating point package (FPAL-96).

| MAJOR I/O FUNCTIONS    |  |
|------------------------|--|
| High Speed Input Unit  | Provides automatic recording of events   |
| High Speed Output Unit | Provides automatic triggering of events and real-time interrupts                   |
| Pulse Width Modulation | Output to drive motors or analog circuits  |
| A to D converter       | Provides analog input  |
| Watchdog Timer         | Resets 8096 if a malfunction occurs  |
| Serial port            | Provides synchronous or asynchronous link  |
| Standard I/O Lines     | Provide interface to the external world when other special features are not needed |

Figure 2-4. Major I/O Functions

Figure 2-5. Instruction Summary

| Mnemonic       | Operands | Operation (Note 1)  | Flags |   |   |   |    |    | Notes |
|----------------|----------|---|-------|---|---|---|----|----|-------|
|                |          |   | Z     | N | C | V | VT | ST |       |
| ADD/ADDB       | 2        | $D \leftarrow D + A$  | ✓     | ✓ | ✓ | ✓ | ↑  | —  |       |
| ADD/ADDB       | 3        | $D \leftarrow B + A$  | ✓     | ✓ | ✓ | ✓ | ↑  | —  |       |
| ADDC/ADDCB     | 2        | $D \leftarrow D + A + C$  | ↓     | ✓ | ✓ | ✓ | ↑  | —  |       |
| SUB/SUBB       | 2        | $D \leftarrow D - A$  | ✓     | ✓ | ✓ | ✓ | ↑  | —  |       |
| SUB/SUBB       | 3        | $D \leftarrow B - A$  | ✓     | ✓ | ✓ | ✓ | ↑  | —  |       |
| SUBC/SUBCB     | 2        | $D \leftarrow D - A + C - 1$  | ↓     | ✓ | ✓ | ✓ | ↑  | —  |       |
| CMP/CMPB       | 2        | $D - A$   | ✓     | ✓ | ✓ | ✓ | ↑  | —  |       |
| MUL/MULU       | 2        | $D, D + 2 \leftarrow D * A$   | —     | — | — | — | —  | ?  | 2     |
| MUL/MULU       | 3        | $D, D + 2 \leftarrow B * A$   | —     | — | — | — | —  | ?  | 2     |
| MULB/MULUB     | 2        | $D, D + 1 \leftarrow D * A$   | —     | — | — | — | —  | ?  | 3     |
| MULB/MULUB     | 3        | $D, D + 1 \leftarrow B * A$   | —     | — | — | — | —  | ?  | 3     |
| DIV/DIVU       | 2        | $D \leftarrow (D, D + 2)/A$<br>D + 2 remainder  | —     | — | — | ✓ | ↑  | —  | 2     |
| DIVB/DIVUB     | 2        | $D \leftarrow (D, D + 1)/A$<br>D + 1 remainder  | —     | — | — | ✓ | ↑  | —  | 3     |
| AND/ANDB       | 2        | $D \leftarrow D \text{ and } A$   | ✓     | ✓ | 0 | 0 | —  | —  |       |
| AND/ANDB       | 3        | $D \leftarrow B \text{ and } A$   | ✓     | ✓ | 0 | 0 | —  | —  |       |
| OR/ORB         | 2        | $D \leftarrow D \text{ or } A$  | ✓     | ✓ | 0 | 0 | —  | —  |       |
| XOR/XORB       | 2        | $D \leftarrow D \text{ (excl. or) } A$  | ✓     | ✓ | 0 | 0 | —  | —  |       |
| LD/LDB         | 2        | $D \leftarrow A$  | —     | — | — | — | —  | —  |       |
| ST/STB         | 2        | $A \leftarrow D$  | —     | — | — | — | —  | —  |       |
| LDBSE          | 2        | $D \leftarrow A; D + 1 \leftarrow \text{SIGN}(A)$   | —     | — | — | — | —  | —  | 3,4   |
| LDBZE          | 2        | $D \leftarrow A; D + 1 \leftarrow 0$  | —     | — | — | — | —  | —  | 3,4   |
| PUSH           | 1        | $SP \leftarrow SP - 2; (SP) \leftarrow A$   | —     | — | — | — | —  | —  |       |
| POP            | 1        | $A \leftarrow (SP); SP \leftarrow SP + 2$   | —     | — | — | — | —  | —  |       |
| PUSHF          | 0        | $SP \leftarrow SP - 2; (SP) \leftarrow \text{PSW};$<br>$\text{PSW} \leftarrow 0000\text{H}$<br>$I \leftarrow 0$ | 0     | 0 | 0 | 0 | 0  | 0  |       |
| POPF           | 0        | $\text{PSW} \leftarrow (SP); SP \leftarrow SP + 2; I \leftarrow \sqrt{\phantom{x}}$                             | ✓     | ✓ | ✓ | ✓ | ✓  | ✓  |       |
| SJMP           | 1        | $PC \leftarrow PC + 11\text{-bit offset}$   | —     | — | — | — | —  | —  | 5     |
| LJMP           | 1        | $PC \leftarrow PC + 16\text{-bit offset}$   | —     | — | — | — | —  | —  | 5     |
| BR(indirect)   | 1        | $PC \leftarrow (A)$   | —     | — | — | — | —  | —  |       |
| SCALL          | 1        | $SP \leftarrow SP - 2; (SP) \leftarrow PC;$<br>$PC \leftarrow PC + 11\text{-bit offset}$                        | —     | — | — | — | —  | —  | 5     |
| LCALL          | 1        | $SP \leftarrow SP - 2; (SP) \leftarrow PC;$<br>$PC \leftarrow PC + 16\text{-bit offset}$                        | —     | — | — | — | —  | —  | 5     |
| RET            | 0        | $PC \leftarrow (SP); SP \leftarrow SP + 2$  | —     | — | — | — | —  | —  |       |
| J(conditional) | 1        | $PC \leftarrow PC + 8\text{-bit offset}$  | —     | — | — | — | —  | —  | 5     |
| JC             | 1        | Jump if C = 1   | —     | — | — | — | —  | —  | 5     |
| JNC            | 1        | Jump if C = 0   | —     | — | — | — | —  | —  | 5     |

**Note**

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
2. D, D + 2 are consecutive WORDS in memory; D is DOUBLE-WORD aligned.
3. D, D + 1 are consecutive BYTES in memory; D is WORD aligned.
4. Changes a byte to a word.
5. Offset is a 2's complement number.

| Mnemonic         | Operands | Operation (Note 1)                                 | Flags |   |   |   |    |    | Notes |
|------------------|----------|--|-------|---|---|---|----|----|-------|
|                  |          |  | Z     | N | C | V | VT | ST |       |
| JE               | 1        | Jump if Z = 1                                      | —     | — | — | — | —  | —  | 5     |
| JNE              | 1        | Jump if Z = 0                                      | —     | — | — | — | —  | —  | 5     |
| JGE              | 1        | Jump if N = 0                                      | —     | — | — | — | —  | —  | 5     |
| JLT              | 1        | Jump if N = 1                                      | —     | — | — | — | —  | —  | 5     |
| JGT              | 1        | Jump if N = 0 and Z = 0                            | —     | — | — | — | —  | —  | 5     |
| JLE              | 1        | Jump if N = 1 or Z = 1                             | —     | — | — | — | —  | —  | 5     |
| JH               | 1        | Jump if C = 1 and Z = 0                            | —     | — | — | — | —  | —  | 5     |
| JNH              | 1        | Jump if C = 0 or Z = 1                             | —     | — | — | — | —  | —  | 5     |
| JV               | 1        | Jump if V = 1                                      | —     | — | — | — | —  | —  | 5     |
| JNV              | 1        | Jump if V = 0                                      | —     | — | — | — | —  | —  | 5     |
| JVT              | 1        | Jump if VT = 1; Clear VT                           | —     | — | — | — | 0  | —  | 5     |
| JNVT             | 1        | Jump if VT = 0; Clear VT                           | —     | — | — | — | 0  | —  | 5     |
| JST              | 1        | Jump if ST = 1                                     | —     | — | — | — | —  | —  | 5     |
| JNST             | 1        | Jump if ST = 0                                     | —     | — | — | — | —  | —  | 5     |
| JBS              | 3        | Jump if Specified Bit = 1                          | —     | — | — | — | —  | —  | 5.6   |
| JBC              | 3        | Jump if Specified Bit = 0                          | —     | — | — | — | —  | —  | 5.6   |
| DJNZ             | 1        | D ← D - 1; if D ≠ 0 then<br>PC ← PC + 8-bit offset | —     | — | — | — | —  | —  | 5     |
| DEC/DECB         | 1        | D ← D - 1  | ✓     | ✓ | ✓ | ✓ | ↑  | —  |       |
| NEG/NEGB         | 1        | D ← 0 - D  | ✓     | ✓ | ✓ | ✓ | ↑  | —  |       |
| INC/INCB         | 1        | D ← D + 1  | ✓     | ✓ | ✓ | ✓ | ↑  | —  |       |
| EXT              | 1        | D ← D; D + 2 ← Sign (D)                            | ✓     | ✓ | 0 | 0 | —  | —  | 2     |
| EXTB             | 1        | D ← D; D + 1 ← Sign (D)                            | ✓     | ✓ | 0 | 0 | —  | —  | 3     |
| NOT/NOTB         | 1        | D ← Logical Not (D)                                | ✓     | ✓ | 0 | 0 | —  | —  |       |
| CLR/CLRB         | 1        | D ← 0  | 1     | 0 | 0 | 0 | —  | —  |       |
| SHL/SHLB/SHLL    | 2        | C ← msb — — — — lsb ← 0                            | ✓     | ? | ✓ | ✓ | ↑  | —  | 7     |
| SHR/SHRB/SHRL    | 2        | 0 → msb — — — — lsb → C                            | ✓     | 0 | ✓ | 0 | —  | ✓  | 7     |
| SHRA/SHRAB/SHRAL | 2        | msb → msb — — — — lsb → C                          | ✓     | ✓ | ✓ | 0 | —  | ✓  | 7     |
| SETC             | 0        | C ← 1  | —     | — | 1 | — | —  | —  |       |
| CLRC             | 0        | C ← 0  | —     | — | 0 | — | —  | —  |       |
| CLRVT            | 0        | VT ← 0   | —     | — | — | — | 0  | —  |       |
| RST              | 0        | PC ← 2080H   | 0     | 0 | 0 | 0 | 0  | 0  | 8     |
| DI               | 0        | Disable All Interrupts (I ← 0)                     | —     | — | — | — | —  | —  |       |
| EI               | 0        | Enable All Interrupts (I ← 1)                      | —     | — | — | — | —  | —  |       |
| NOP              | 0        | PC ← PC + 1  | —     | — | — | — | —  | —  |       |
| SKIP             | 0        | PC ← PC + 2  | —     | — | — | — | —  | —  |       |
| NORML            | 2        | Normalize  | ✓     | 1 | 0 | — | —  | —  | 7     |
| TRAP             | 0        | SP ← SP - 2; (SP) ← PC<br>PC ← (2010H)             | —     | — | — | — | —  | —  | 9     |

**Note**

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
5. Offset is a 2's complement number.
6. Specified bit is one of the 2048 bits in the register file.
7. The "L" (Long) suffix indicates double-word operation.
8. Initiates a Reset by pulling RESET low. Software should re-initialize all the necessary registers with code starting at 2080H.
9. The assembler will not accept this mnemonic.

One operand of most of the instructions can be used with any one of six addressing modes. These modes increase the flexibility and overall execution speed of the 8096. The addressing modes are: register-direct, immediate, indirect, indirect with auto-increment, and long and short indexed.

The fastest instruction execution is gained by using either register direct or immediate addressing. Register-direct addressing is similar to normal direct addressing, except that only addresses in the register file or SFRs can be addressed. The indexed mode is used to directly address the remainder of the 64K address space. Immediate addressing operates as it would be expected, using the data following the opcode as the operand.

Both of the indirect addressing modes use the value in a word register as the address of the operand. If the indirect auto-increment mode is used then the word register is incremented by one after a byte access or by two after a word access. This mode is particularly useful for accessing lookup tables.

Access to any of the locations in the 64K address space can be obtained by using the long indexed

addressing mode. In this mode a 16-bit 2's complement value is added to the contents of a word register to form the address of the operand. By using the zero register as the index, ASM96 (the assembler) can accept "direct" addressing to any location. The zero register is located at 0000H and always has a value of zero. A short indexed mode is also available to save some time and code. This mode uses an 8-bit 2's complement number as the offset instead of a 16-bit number.

### 2.2.2. Assembly language

The multiple addressing modes of the 8096 make it easy to program in assembly language and provide an excellent interface to high level languages. The instructions accepted by the assembler consist of mnemonics followed by either addresses or data. A list of the mnemonics and their functions are shown in Figure 2-5. The addresses or data are given in different formats depending on the addressing mode. These modes and formats are shown in Figure 2-6.

Additional information on 8096 assembly language is available in the MCS-96 Macro Assembler Users Guide, listed in the bibliography.

|   |                         |   |
|---|-------------------------|---|
| Mnem  | Dest or Src1            | : One operand direct                    |
| Mnem  | Dest, Src1              | : Two operand direct                    |
| Mnem  | Dest, Src1, Src2        | : Three operand direct                  |
| Mnem  | #Src1                   | : One operand immediate                 |
| Mnem  | Dest, #Src1             | : Two operand immediate                 |
| Mnem  | Dest, Src1, #Src2       | : Three operand immediate               |
| Mnem  | [addr]                  | : One operand indirect                  |
| Mnem  | [addr] +                | : One operand indirect auto-increment   |
| Mnem  | Dest, [addr]            | : Two operand indirect                  |
| Mnem  | Dest, [addr] +          | : Two operand indirect auto-increment   |
| Mnem  | Dest, Src1, [addr]      | : Three operand indirect                |
| Mnem  | Dest, Src1, [addr] +    | : Three operand indirect auto-increment |
| Mnem  | Dest, offs [addr]       | : Two operand indexed (short or long)   |
| Mnem  | Dest, Src1, offs [addr] | : Three operand indexed (short or long) |
| Where: "Mnem" is the instruction mnemonic   |                         |   |
| "Dest" is the destination register  |                         |   |
| "Src1", "Src2" are the source registers   |                         |   |
| "addr" is a register containing a value to be used in computing the address of an operand |                         |   |
| "offs" is an offset used in computing the address of an operand                           |                         |   |

Figure 2-6. Instruction Format



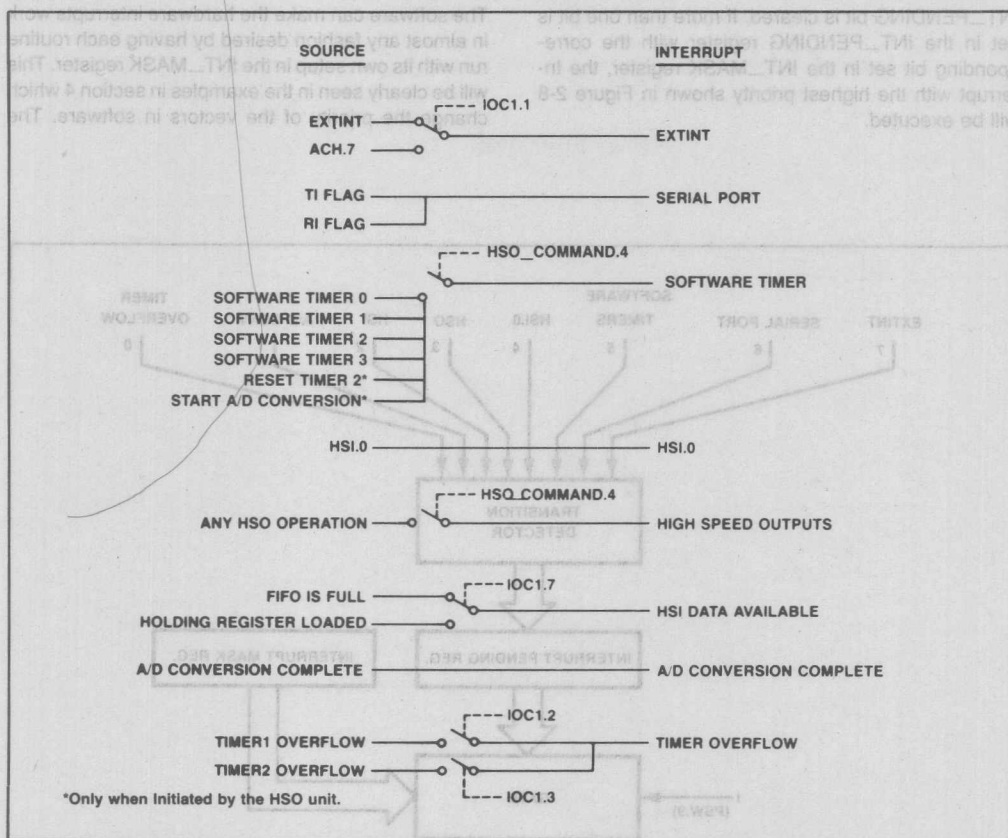


Figure 2-7. Interrupt Sources

### 2.2.3. Interrupts

The flexibility of the instruction set is carried through into the interrupt system. There are 20 different interrupt sources that can be used on the 8096. The 20 sources vector through 8 locations or interrupt vectors. The vector names and their sources are shown in Figure 2-7, with their locations listed in Figure 2-8. Control of the interrupts is handled through the Interrupt Pending Register (INT\_PENDING), the Interrupt Mask Register (INT\_MASK), and the I bit in the PSW (PSW.9). Figure 2-9 shows a block diagram of the interrupt structure. The INT\_PENDING register contains bits which get set by hardware when an interrupt occurs. If the interrupt mask register bit for that source is a 1 and PSW.9=1, a vector will be taken to the address listed in the interrupt vector table for that source. When the vector is taken the

| Source                  | Vector Location |            | Priority       |
|-------------------------|-----------------|------------|----------------|
|                         | (High Byte)     | (Low Byte) |                |
| Software                | 2011H           | 2010H      | Not Applicable |
| Extint                  | 200FH           | 200EH      |                |
| Serial Port             | 200DH           | 200CH      | 6              |
| Software Timers         | 200BH           | 200AH      | 5              |
| HSI.0                   | 2009H           | 2008H      | 4              |
| High Speed Outputs      | 2007H           | 2006H      | 3              |
| HSI Data Available      | 2005H           | 2004H      | 2              |
| A/D Conversion Complete | 2003H           | 2002H      | 1              |
| Timer Overflow          | 2001H           | 2000H      | 0 (Lowest)     |

Figure 2-8. Interrupt Vectors and Priorities

INT\_PENDING bit is cleared. If more than one bit is set in the INT\_PENDING register with the corresponding bit set in the INT\_MASK register, the Interrupt with the highest priority shown in Figure 2-8 will be executed.

The software can make the hardware interrupts work in almost any fashion desired by having each routine run with its own setup in the INT\_MASK register. This will be clearly seen in the examples in section 4 which change the priority of the vectors in software. The

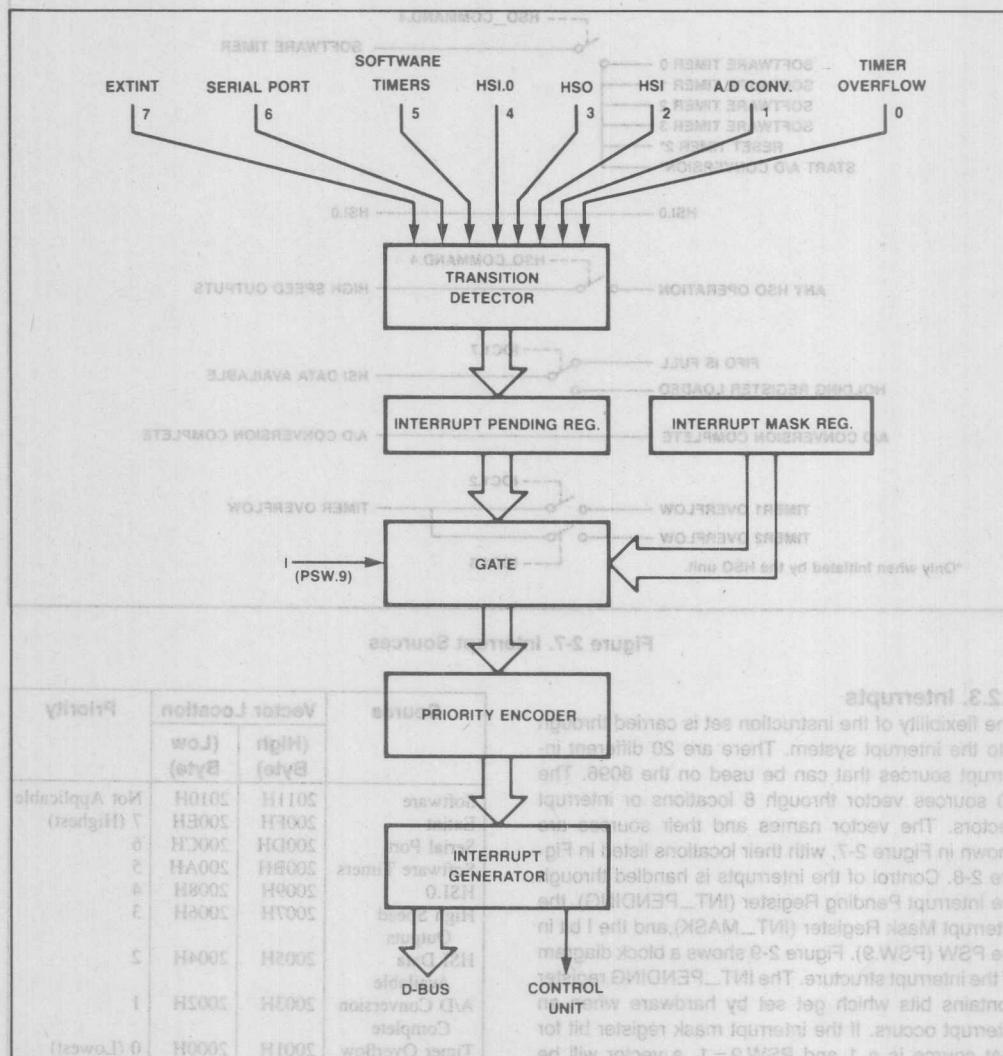


Figure 2-9. Interrupt Structure Block Diagram

|    |    |    |    |    |    |    |    |          |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|----------|----|----|----|----|----|----|----|
| 15 | 14 | 13 | 12 | 11 | 10 | 09 | 08 | 07       | 06 | 05 | 04 | 03 | 02 | 01 | 00 |
| Z  | N  | V  | VT | C  | —  | I  | ST | INT_MASK |    |    |    |    |    |    |    |

**WHERE:**

**Z** IS THE ZERO FLAG. IT IS SET WHEN THE RESULT OF AN OPERATION IS ZERO.

**N** IS THE NEGATIVE FLAG. IT IS SET TO THE ALGEBRAICALLY CORRECT SIGN OF THE RESULT REGARDLESS OF OVERFLOWS.

**V** IS THE OVERFLOW FLAG. IT IS SET IF AN OVERFLOW OCCURS.

**VT** IS THE OVERFLOW TRAP FLAG. IT IS SET WHEN THE VT FLAG IS SET AND CLEARED BY JVT, JNVT, OR CLRVT.

**C** IS THE CARRY FLAG. IT IS SET IF A CARRY WAS GENERATED BY THE PRIOR OPERATION.

**I** IS THE GLOBAL INTERRUPT ENABLE BIT.

**ST** IS THE STICKY BIT. IT IS SET DURING A RIGHT SHIFT IF A ONE WAS SHIFTED INTO AND THEN OUT OF THE CARRY FLAG.

**INT\_MASK** IS THE INTERRUPT MASK REGISTER AND CONTAINS BITS WHICH INDIVIDUALLY ENABLE THE 8 INTERRUPT VECTORS.

Figure 2-10. The PSW Register

PSW (shown in Figure 2-10), stores the INT\_MASK register in its lower byte so that the mask register can be pushed and popped along with the machine status when moving in and out of routines. The action of pushing flags clears the PSW which includes PSW.9, the interrupt enable bit. Therefore, after a PUSHF instruction interrupts are disabled. In most cases an interrupt service routine will have the basic structure shown below.

**INT VECTOR:**

```

PUSHF
LDB INT_MASK, #xxxxxxxB
EI
; Insert service routine here
POPF
RET

```

The PUSHF instruction saves the PSW including the old INT\_MASK register. The PSW, including the interrupt enable bit are left cleared. If some interrupts need to be enabled while the service routine runs, the INT\_MASK is loaded with a new value and interrupts are globally enabled before the service routine continues. At the end of the service routine a

POPF instruction is executed to restore the old PSW. The RET instruction is executed and the code returns to the desired location. Although the POPF instruction can enable the interrupts the next instruction will always execute. This prevents unnecessary building of the stack by ensuring that the RET always executes before another interrupt vector is taken.

## 2.3. ON-CHIP I/O SECTION

All of the on-chip I/O features of the 8096 can be accessed through the special function registers, as shown in Figure 2-3. The advantage of using register-mapped I/O is that these registers can be used as the sources or destinations of CPU operations. There are seven major I/O functions. Each one of these will be considered with a section of code to exemplify its usage. The first section covered will be the High Speed I/O (HSIO), subsystem. This section includes the High Speed Input (HSI) unit, High Speed Output (HSO) unit, and the Timer/Counter section.

### 2.3.1. Timer/Counters

The 8096 has two time bases, Timer1 and Timer2. Timer1 is a 16-bit free running timer which is incremented every 8 state times. (A state time is 3 oscillator periods, or 0.25 microseconds with a 12 MHz







messages to various devices to have them turn on, turn off, start processing, or reset. Since the programmed times can be referenced to either Timer 1 or Timer 2, the HSO makes the two timers look like many. For example, if several events have to occur at specific times, the HSO unit can schedule all of the events based on a single timer. The events that can be scheduled to occur and the format of the command written to the HSO Command register are shown in Figure 2-13.

The software timers listed in the figure are actually 4 software flags in I/O Status Register 1 (IOS1). These flags can be set, and optionally cause an interrupt, at any time based on Timer 1 or Timer 2. In most cases these timers are used to trigger interrupt routines which must occur at regular intervals. A multi-task process can easily be set up using the software timers.

A CAM (Content Addressable Memory) file is the main component of the HSO. This file stores up to eight events which are pending to occur. Every state time one location of the CAM is compared the two timers. After 8 state times, (two microseconds with a 12 MHZ clock), the entire CAM has been searched for time matches. If a match occurs the specified event will be triggered and that location of the CAM will be made available for another pending event. A block diagram of the HSO unit is shown in Figure 2-14.

#### 2.3.4. Serial Port

Controlling a device from a remote location is a simple task that frequently requires additional hardware with many processors. The 8096 has an on-chip serial port to reduce the total number of chips required in the system. The serial port is similar to that on the MCS-51 product line. It has one synchronous and

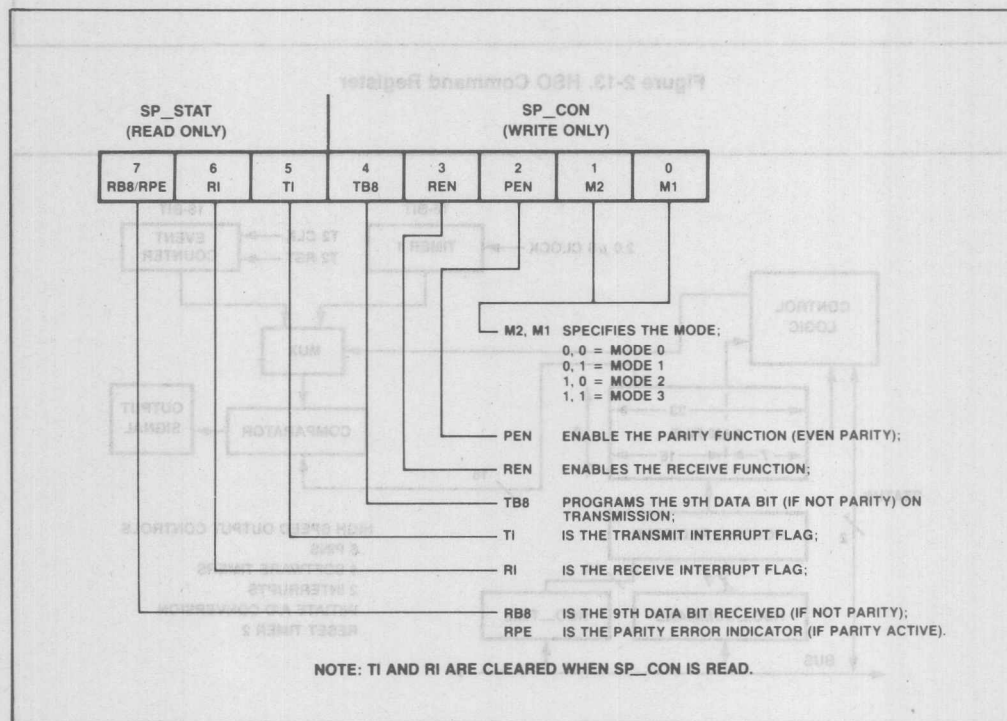


Figure 2-15. Serial Port Control/Status Register

three asynchronous modes. In the asynchronous modes baud rates of up to 187.5 Kbaud can be used, while in the synchronous mode rates up to 1.5 Mbaud are available. The chip has a baud rate generator which is independent of Timer 1 and Timer 2, so using the serial port does not take away any of the HSI, HSO or timer flexibility or functionality.

Control of the serial port is provided through the SPCON/SPSTAT (Serial Port CONTROL/Serial Port STATUS) register. This register, shown in Figure 2-15, has some bits which are read only and others which are write only. Although the functionality of the port is similar to that of the 8051, the names of some of the modes and control bits are different. The way in which the port is used from a software standpoint is also slightly different since RI and TI are cleared after each read of the register.

The four modes of the serial port are referred to as modes 0, 1, 2, and 3. Mode 0 is the synchronous mode, and is commonly used to interface to shift registers for I/O expansion. In this mode the port outputs a pulse train on the TXD pin and either transmits or receives data on the RXD pin. Mode 1 is the standard asynchronous mode, 8 bits plus a stop and start bit are sent or received. Modes 2 and 3 handle 9 bits plus a stop and start bit. The difference between the two is, that in Mode 2 the serial port interrupt will not be activated unless the ninth data bit is a one; in Mode 3 the interrupt is activated whenever a byte is received. These two modes are commonly used for interprocessor communication.

Using XTAL1:

$$\text{Mode 0: Baud Rate} = \frac{\text{XTAL1 frequency}}{4*(B+1)}; B \neq 0$$

$$\text{Others: Baud Rate} = \frac{\text{XTAL1 frequency}}{64*(B+1)}$$

Using T2CLK:

$$\text{Mode 0: Baud Rate} = \frac{\text{T2CLK frequency}}{B}; B \neq 0$$

$$\text{Others: Baud Rate} = \frac{\text{T2CLK frequency}}{16*B}; B \neq 0$$

Note that B cannot equal 0, except when using XTAL1 in other than mode 0.

Figure 2-16. Baud Rate Formulas

Baud rates for all of the modes are controlled through the Baud Rate register. This is a byte wide register which is loaded sequentially with two bytes, and internally stores the value as a word. The least significant byte is loaded to the register followed by the most significant. The most significant bit of the baud value determines the clock source for the baud rate generator. If the bit is a one, the XTAL1 pin is used as the source, if it is a zero, the T2 CLK pin is used. The formulas shown in Figure 2-16 can be used to calculate the baud rates. The variable "B" is used to represent the least significant 15 bits of the value loaded into the baud rate register.

The baud rate register values for common baud rates are shown in Figure 2-17. These values can be used when XTAL1 is selected as the clock source for serial modes other than Mode 0. The percentage deviation from theoretical is listed to help assess the reliability of a given setup. In most cases a serial link will work if there is less than a 2.5% difference between the baud rates of the two systems. This is based on the assumption that 10 bits are transmitted per frame and the last bit of the frame must be valid for at least six-eighths of the bit time. If the two systems deviate from theoretical by 1.25% in opposite directions the maximum tolerance of 2.5% will be reached. Therefore, caution must be used when the baud rate deviation approaches 1.25% from theoretical. Note that an XTAL1 frequency of 11.0592 MHz can be used with the table values for 11 MHz to provide baud rates that have 0.0 percent deviation from theoretical. In most applications, however, the accuracy available when using an 11 MHz input frequency is sufficient.

Serial port Mode 1 is the easiest mode to use as there is little to worry about except initialization and loading and unloading SBUF, the Serial port Buffer. If parity is enabled, (ie. PEN1), 7 bits plus even parity are used instead of 8 data bits. The parity calculation is done in hardware for even parity. Modes 2 and 3 are similar to Mode 1, except that the ninth bit needs to be controlled and read. It is also not possible to enable parity in Mode 2. When parity is enabled in Mode 3 the ninth bit becomes the parity bit. If parity is not enabled, (ie. PEN = 0), the TB8 bit controls the state of the ninth transmitted bit. This bit must be set prior to each transmission. On reception, if PEN = 0, the RB8 bit indicates the state of the ninth received bit. If parity is enabled, (ie. PEN = 1), the same bit is called RPE (Receive Parity Error), and is used to indicate a parity error.

| XTAL1 Frequency = 12.0 MHz |                     |               |
|----------------------------|---------------------|---------------|
| Baud Rate                  | Baud Register Value | Percent Error |
| 19.2K                      | 8009H               | +2.40         |
| 9600                       | 8013H               | +2.40         |
| 4800                       | 8026H               | -0.16         |
| 2400                       | 804DH               | -0.16         |
| 1200                       | 809BH               | +0.16         |
| 300                        | 8270H               | 0.00          |
| XTAL1 Frequency = 11.0 MHz |                     |               |
| Baud Rate                  | Baud Register Value | Percent Error |
| 19.2K                      | 8008H               | +0.54         |
| 9600                       | 8011H               | +0.54         |
| 4800                       | 8023H               | +0.54         |
| 2400                       | 8047H               | +0.54         |
| 1200                       | 808EH               | -0.16         |
| 300                        | 823CH               | +0.01         |
| XTAL1 Frequency = 10.0 MHz |                     |               |
| Baud Rate                  | Baud Register Value | Percent Error |
| 19.2K                      | 8007H               | -1.70         |
| 9600                       | 800FH               | -1.70         |
| 4800                       | 8020H               | +1.38         |
| 2400                       | 8040H               | -0.16         |
| 1200                       | 8081H               | -0.16         |
| 300                        | 8208H               | +0.03         |

Figure 2-17. Baud Rate Values for 10,11,12 MHz

The software used to communicate between processors is simplified by making use of Modes 2 and 3. In a basic protocol the ninth bit is called the address bit. If it is set high then the information in that byte is either the address of one of the processors on the link, or a command for all the processors. If the bit is a zero, the byte contains information for the processor or processors previously addressed. In standby mode all processors wait in Mode 2 for a byte with the address bit set. When they receive that byte, the software determines if the next message is for them. The processor that is to receive the message switches to

Mode 3 and receives the information. Since this information is sent with the ninth bit set to zero, none of the processors set to Mode 2 will be interrupted. By using this scheme the overall CPU time required for the serial port is minimized.

A typical connection diagram for the multi-processor mode is shown in Figure 2-18. This type of communication can be used to connect peripherals to a desk top computer, the axis of a multi-axis machine, or any other group of microcontrollers jointly performing a task.



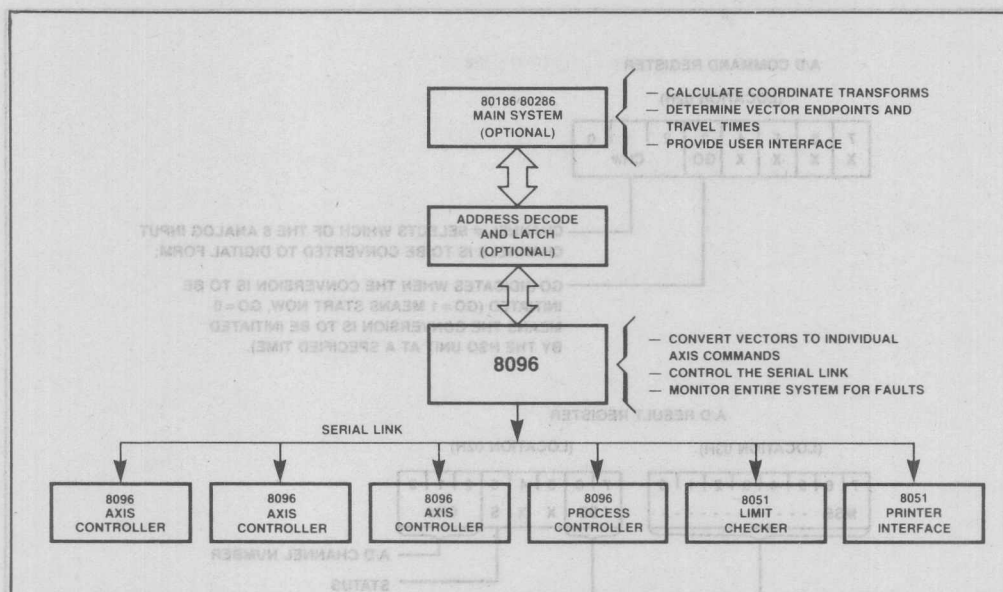


Figure 2-18. Multiprocessor Communication

Mode 0, the synchronous mode, is typically used for interfacing to shift registers for I/O expansion. The software to control this mode involves the REN (Receiver ENable) bit, the clearing of the RI bit, and writing to SBUF. To transmit to a shift register, REN is set to zero and SBUF is loaded with the information. The information will be sent and then the TI flag will be set. There are two ways to cause a reception to begin. The first is by causing a rising edge to occur on the REN bit, the second is by clearing RI with REN=1. In either case, RI is set again when the received byte is available in SBUF.

### 2.3.5. A to D Converter

Analog inputs are frequently required in a microcontroller application. The 8097 has a 10-bit A to D converter that can use any one of eight input channels. The conversions are done using the successive approximation method, and require 168 state times (42 microseconds with a 12 MHz clock.)

The results are guaranteed monotonic by design of the converter. This means that if the analog input voltage changes, even slightly, the digital value will either stay the same or change in the same direction as the analog input. When doing process control al-

gorithms, it is frequently the changes in inputs that are required, not the absolute accuracy of the value. For this reason, even if the absolute accuracy of a 10-bit converter is the same as that of an 8-bit converter, the 10-bit monotonic converter is much more useful.

Since most of the analog inputs which are monitored by a microcontroller change very slowly relative to the 42 microsecond conversion time, it is acceptable to use a capacitive filter on each input instead of a sample and hold. The 8097 does not have an internal sample and hold, so it is necessary to ensure that the input signal does not change during the conversion time. The input to the A/D must be between ANGND and VREF. ANGND must be within a few millivolts of VSS and VREF must be within a few tenths of a volt of VCC.

Using the A to D converter on the 8097 can be a very low software overhead task because of the interrupt and HSO unit structure. The A to D can be started by the HSO unit at a preset time. When the conversion is complete it is possible to generate an interrupt. By using these features the A to D can be run under complete interrupt control. The A to D can also be

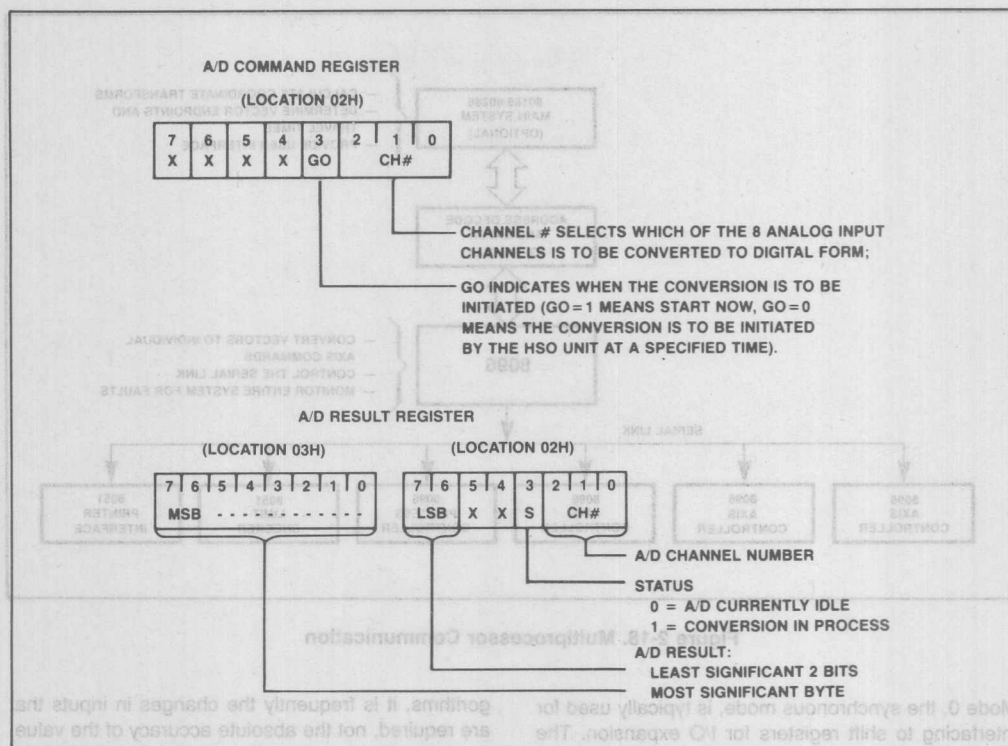


Figure 2-19. A to D Result/Command Register

directly controlled by software flags which are located in the AD\_RESULT/AD\_COMMAND Register, shown in Figure 2-19.

### 2.3.6. PWM Register

Analog outputs are just as important as analog inputs when connecting to a piece of equipment. True digital to analog converters are difficult to make on a microprocessor because of all of the digital noise and the necessity of providing an on chip, relatively high current, rail to rail driver. They also take up a fair amount of silicon area which can be better used for other features. The A to D converter does use a D to A, but the currents involved are very small.

For many applications an analog output signal can be replaced by a Pulse Width Modulated (PWM) signal. This signal can be easily generated in hardware,

and takes up much less silicon area than a true D to A. The signal is a variable duty cycle, fixed frequency waveform that can be integrated to provide an approximation to an analog output. The frequency is fixed at a period of 64 microseconds for a 12 MHz clock speed. Controlling the PWM simply requires writing the desired duty cycle value (an 8-bit value) to the PWM Register. Some typical output waveforms that can be generated are shown in Figure 2-20. Converting the PWM signal to an analog signal varies in difficulty, depending upon the requirements of the system. Some systems, such as motors or switching power supplies actually require a PWM signal, not a true analog one. For many other cases it is necessary only to amplify the signal so that it switches rail-to-rail, and then filter it. Switching rail-to-rail means that the output of the amplifier will be a reference value when the input is a logical one, and the output will

be zero when the input is a logical zero. The filter can be a simple RC network or an active filter. If a large amount of current is needed a buffer is also required. For low output currents, (less than 100 microamps or so), the circuit shown in Figure 2-21 can be used.

The RC network determines how quiet the output is, but the quieter the output, the slower it can change. The design of high accuracy voltage followers and active filters is beyond the scope of this paper, however many books on the subject are available.

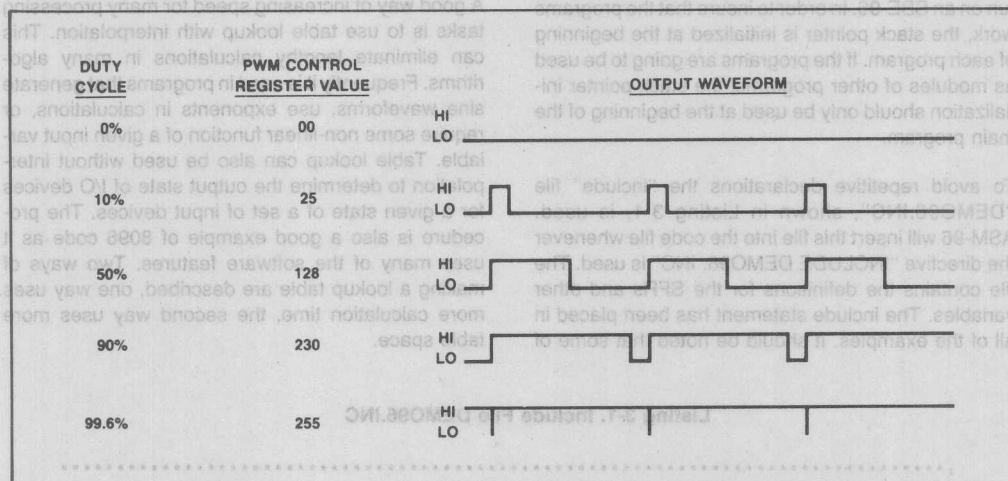


Figure 2-20. PWM Output Waveforms

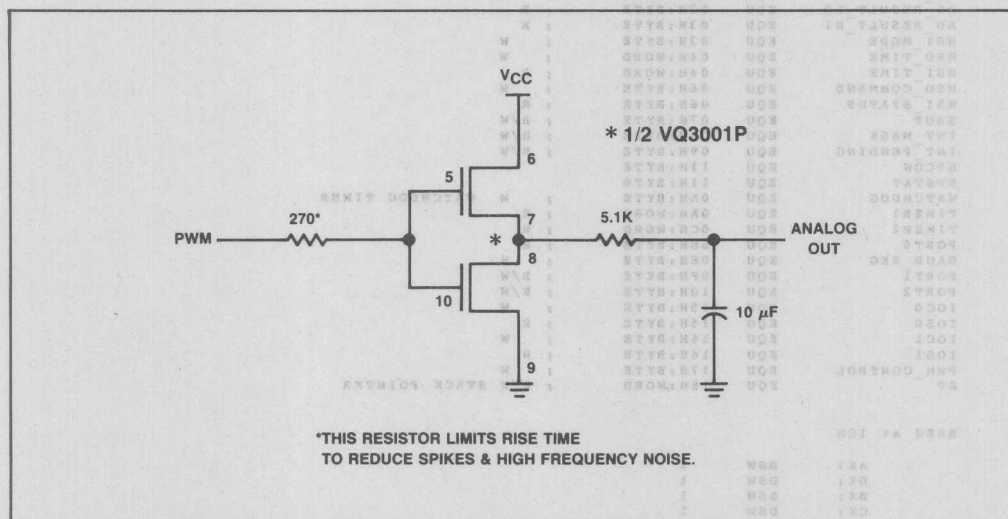


Figure 2-21. PWM to Analog Conversion Circuitry

### 3.0 BASIC SOFTWARE EXAMPLES

The examples in this section show how to use each I/O feature individually. Examples of using more than one feature at a time are described in section 4. All of the examples in this ap-note are set up to be used as listed. If run through ASM-96 they will load and run on an SBE-96. In order to insure that the programs work, the stack pointer is initialized at the beginning of each program. If the programs are going to be used as modules of other programs, the stack pointer initialization should only be used at the beginning of the main program.

To avoid repetitive declarations the "include" file "DEMO96.INC", shown in Listing 3-1, is used. ASM-96 will insert this file into the code file whenever the directive "INCLUDE DEMO96.INC" is used. The file contains the definitions for the SFRs and other variables. The include statement has been placed in all of the examples. It should be noted that some of

the labels in this file are different from those in the file 8096.INC that is provided in the ASM-96 package.

### 3.1. USING THE 8096'S PROCESSING SECTION

#### 3.1.1. Table Interpolation

A good way of increasing speed for many processing tasks is to use table lookup with interpolation. This can eliminate lengthy calculations in many algorithms. Frequently it is used in programs that generate sine waveforms, use exponents in calculations, or require some non-linear function of a given input variable. Table lookup can also be used without interpolation to determine the output state of I/O devices for a given state of a set of input devices. The procedure is also a good example of 8096 code as it uses many of the software features. Two ways of making a lookup table are described, one way uses more calculation time, the second way uses more table space.

Listing 3-1. Include File DEMO96.INC

```

; *****
;
; DEMO96.INC - DEFINITION OF SYMBOLIC NAMES FOR THE I/O REGISTERS OF THE 8096
;
; *****
;
ZERO EQU 00h:WORD ; R/W
AD_COMMAND EQU 02h:BYTE ; W
AD_RESULT_LO EQU 02h:BYTE ; R
AD_RESULT_HI EQU 03h:BYTE ; R
HSI_MODE EQU 03h:BYTE ; W
HSO_TIME EQU 04h:WORD ; W
HSI_TIME EQU 04h:WORD ; R
HSO_COMMAND EQU 06h:BYTE ; W
HSI_STATUS EQU 06h:BYTE ; R
SBUF EQU 07h:BYTE ; R/W
INT_MASK EQU 08h:BYTE ; R/W
INT_PENDING EQU 09h:BYTE ; R/W
SPCON EQU 11h:BYTE
SPSTAT EQU 11h:BYTE
WATCHDOG EQU 0Ah:BYTE ; W WATCHDOG TIMER
TIMER1 EQU 0Ah:WORD ; R
TIMER2 EQU 0Ch:WORD ; R
PORT0 EQU 0Eh:BYTE ; R
BAUD_REG EQU 0Eh:BYTE ; W
PORT1 EQU 0Fh:BYTE ; R/W
PORT2 EQU 10h:BYTE ; R/W
IOC0 EQU 15h:BYTE ; W
IOS0 EQU 15h:BYTE ; R
IOC1 EQU 16h:BYTE ; W
IOS1 EQU 16h:BYTE ; R
PWM_CONTROL EQU 17h:BYTE ; W
SP EQU 18h:WORD ; R/W STACK POINTER

```

RSEG at 1CH

```

AX: DSW 1
DX: DSW 1
BX: DSW 1
CX: DSW 1

```

```

AL EQU AX : BYTE
AH EQU (AX+1) : BYTE

```



In both methods the procedure is similar. Values of a function are stored in memory for specific input values. To compute the output function for an input that is not listed, a linear approximation is made based on the nearest inputs and nearest outputs. As an example, consider the table below.

If the input value was one of those listed then there would be no problem. Unfortunately the real world is never so kind. The input number will probably be 259 or something similar. If this is the case linear interpolation would provide a reasonable result. The formula is:

$$\text{Delta Out} = \frac{\text{Upper Output} - \text{Lower Output}}{\text{Upper Input} - \text{Lower Input}} * (\text{Actual Input} - \text{Lower Input})$$

$$\text{Actual Output} = \text{Lower Output} + \text{Delta Out}$$

For the value of 259 the solution is:

$$\text{Delta Out} = \frac{900-400}{300-200} * (259-200) = \frac{500}{100} * 59 = 5 * 59 = 295$$

$$\text{Actual Output} = 200 + 295 = 495$$

To make the algorithm easier, (and therefore faster), it is appropriate to limit the range and accuracy of the function to only what is needed. It is also advantageous to make the input step (Upper Input-Lower Input) equal to a power of 2. This allows the substitution of multiple right shifts for a divide operation, thus speeding up throughput. The 8096 allows multiple arithmetic right shifts with a single instruction providing a very fast divide if the divisor is a power of two.

For the purpose of an example, a program with a 12-bit output and an 8-bit input has been written. An input step of 16 ( $2^{**}4$ ) was selected. To cover the input range 17 words are needed,  $255/16 + 1$  word to handle values in the last 15 bytes of input range. Although only 12 bits are required for the output, the 16-bit architecture offers no penalty for using 16 instead of 12 bits.

The program for this example, shown in Listing 3-2, uses the definitions and equates from Listing 3-1, only the additional equates and definitions are shown in the code.

| Input Value | Relative Table Address | Table Value |
|-------------|------------------------|-------------|
| 100         | 0001H                  | 100         |
| 200         | 0002H                  | 400         |
| 300         | 0003H                  | 900         |
| 400         | 0004H                  | 1600        |

Listing 3-2. ASM-96 Code for Table Lookup Routine 1

```

$TITLE('INTER1.APT: Interpolation routine 1')
; ; ; ; ; 8096 Assembly code for table lookup and interpolation

$INCLUDE('F1:DEMO96.INC') ; Include demo definitions

RSEG at 22H
    IN_VAL:      dab 1 ; Actual Input Value
    TABLE_LOW: dsb 1
    TABLE_HIGH: dsb 1
    IN_DIF:      dsb 1 ; Upper Input - Lower Input
    IN_DIFB:     equ IN_DIF :byte
    TAB_DIF:     dsb 1 ; Upper Output - Lower Output
    OUT:         dsb 1
    RESULT:      dsb 1
    OUT_DIF:     dsb 1 ; Delta Out
    OUT_DIFB:    dsb 1

CSEG at 2080H
    LD SP, #100H

```

```

look: LDB AL, IN_VAL ; Load temp with Actual Value
      SHRB AL, #3 ; Divide the byte by 8
      ANDB AL, #1111110B ; Insure AL is a word address
      ; This effectively divides AL by 2
      ; so AL = IN_VAL/16
      LDBZ AX, AL ; Load byte AL to word AX
      LD TABLE_LOW, TABLE [AX] ; TABLE_LOW is loaded with the value
      ; in the table at table location AX
      LD TABLE_HIGH, (TABLE+2)[AX] ; TABLE_HIGH is loaded with the
      ; value in the table at table
      ; location AX+2
      ; (The next value in the table)
      SUB TAB_DIF, TABLE_HIGH, TABLE_LOW ; TAB_DIF=TABLE_HIGH-TABLE_LOW
      ; TAB_DIF=least significant 4 bits
      ; of IN_VAL
      LDBZ IN_DIFB, IN_DIFB ; Load byte IN_DIFB to word IN_DIF
      MUL OUT_DIF, IN_DIF, TAB_DIF ; Output_difference =
      ; Input_difference*Table_difference
      SHRAL OUT_DIF, #4 ; Divide by 16 (2**4)
      ADD OUT, OUT_DIF, TABLE_LOW ; Add output difference to output
      ; generated with truncated IN_VAL
      ; as input
      SHRA OUT, #4 ; Round to 12-bit answer
      ; Round up, if Carry = 1
      ADDC OUT, zero
      ; Store OUT to RESULT
      ST OUT, RESULT
      BR look ; Branch to "look:"

```

cseg AT 2100H

|        |     |        |        |        |       |                     |
|--------|-----|--------|--------|--------|-------|---------------------|
| table: | DCW | 0000H, | 2000H, | 3400H, | 4C00H | ; A random function |
|        | DCW | 5D00H, | 6A00H, | 7200H, | 7800H |                     |
|        | DCW | 7B00H, | 7D00H, | 7600H, | 6D00H |                     |
|        | DCW | 5D00H, | 4B00H, | 3400H, | 2200H |                     |
|        | DCW | 1000H, |        |        |       |                     |
| END    |     |        |        |        |       |                     |

If the function is known at the time of writing the software it is also possible to calculate in advance the change in the output function for a given change in the input. This method can save a divide and a few other instructions at the expense of doubling the size

of the lookup table. There are many applications where time is critical and code space is overly abundant. In these cases the code in Listing 3-3 will work to the same specifications as the previous example.

Listing 3-3. ASM-96 Code For Table Lookup Routine 2

```

$TITLE('INTER2.APT: Interpolation routine 2')
; ; ; ; ; 8096 Assembly code for table lookup and interpolation
; ; ; ; ; Using tabled values in place of division
$INCLUDE('F1:DEMO96.INC') ; Include demo definitions
RSEG at 24H
IN_VAL: dsb 1 ; Actual Input Value
TABLE_LOW: dsb 1 ; Table value for function
TABLE_INC: dsb 1 ; Incremental change in function
IN_DIF: dsb 1 ; Upper Input - Lower Input
IN_DIFB: equ IN_DIF :byte
OUT: dsb 1
RESULT: dsb 1
OUT_DIF: dsb 1 ; Delta Out

```

```

CSEG at 2080H
LD      SP, #100H      ; Initialize SP to top of reg. file
look:   LDB      AL, IN_VAL      ; Load temp with Actual Value
        SHRB     AL, #3         ; Divide the byte by 8
        ANDB     AL, #1111110B  ; Insure AL is a word address
        ; This effectively divides AL by 2
        ; so AL = IN_VAL/16
        LDBZE    AX, AL        ; Load byte AL to word AX

        LD       TABLE_LOW, VAL_TABLE[AX] ; TABLE LOW is loaded with the value
        ; in the value table at location AX

        LD       TABLE_INC, INC_TABLE[AX] ; TABLE INC is loaded with the value
        ; in the increment table at
        ; location AX+2

        ANDB     IN_DIFB, IN_VAL, #0FH ; IN_DIFB=least significant 4 bits
        ; of IN_VAL
        LDBZE    IN_DIF, IN_DIFB ; Load byte IN_DIFB to word IN_DIF

        MUL      OUT_DIF, IN_DIF, TABLE_INC ; Output_difference =
        ; Input_difference*Incremental_change

        ADD      OUT, OUT_DIF, TABLE_LOW ; Add output difference to output
        ; generated with truncated IN_VAL
        ; as input
        SHR      OUT, #4         ; Round to 12-bit answer
        ADDC     OUT, zero       ; Round up if Carry = 1

no_inc: ST      OUT, RESULT      ; Store OUT to RESULT
        BR       look          ; Branch to "look:"

cseg    AT 2100H

val_table:
        DCW      0000H, 2000H, 3400H, 4C00H ; A random function
        DCW      5D00H, 6A00H, 7200H, 7800H
        DCW      7B00H, 7D00H, 7600H, 6D00H
        DCW      5D00H, 4B00H, 3400H, 2200H
        DCW      1000H

inc_table:
        DCW      0200H, 0140H, 0180H, 0110H ; Table of incremental
        DCW      00D0H, 0080H, 0060H, 0030H ; differences
        DCW      00020H, 0FF90H, 0FF70H, 0FF00H
        DCW      0FEE0H, 0FE90H, 0FEE0H, 0FEE0H

END

```

By making use of the second lookup table, one word of RAM was saved and 16 state times. In most cases this time savings would not make much of a difference, but when pushing the processor to the limit, microseconds can make or break a design.

### 3.1.2. PL/M-96

Intel provides high level language support for most of its micro processors and microcontrollers in the form of PL/M. Specifically, PL/M refers to a family of languages, each similar in syntax, but specialized for the device for which it generates code. The PL/M syntax is similar to PL/1, and is easy to learn. PLM-96 is the version of PL/M used for the 8096. It is very code efficient as it was written specifically for the MCS-96 family. PLM-96 most closely resembles PLM-86, although it has bit and I/O functions similar to PLM-51. One line of PL/M-code can take the place

of many lines of assembly code. This is advantageous to the programmer, since code can usually be written at a set number of lines per hour, so the less lines of code that need to be written, the faster the task can be completed.

If the first example of interpolation is considered, the PLM-96 code would be written as shown in Listing 3-4. Note that version 1.0 of PLM-96 does not support 32-bit results of 16 by 16 multiplies, so the ASM-96 procedure "DMPY" is used. Procedure DMPY, shown in Listing 3-5, must be assembled and linked with the compiled PLM-96 program using RL-96, the relocater and linker. The command line to be used is:

```

RL96 PLMEX1.OBJ, DMPY.OBJ, PLM96.LIB &
to PLMOUT.OBJ ROM (2080H-3FFFH)

```

Listing 3-4. PLM-96 Code For Table Lookup Routine 1

```

/* PLM-96 CODE FOR TABLE LOOK-UP AND INTERPOLATION */
PLMEX:    DO;

DECLARE IN_VAL      WORD      PUBLIC;
DECLARE TABLE_LOW  INTEGER   PUBLIC;
DECLARE TABLE_HIGH INTEGER   PUBLIC;
DECLARE TABLE_DIF  INTEGER   PUBLIC;
DECLARE OUT_DIF     INTEGER   PUBLIC;
DECLARE RESULT      INTEGER   PUBLIC;
DECLARE OUT_DIF     LONGINT    PUBLIC;
DECLARE TEMP        WORD      PUBLIC;

DECLARE TABLE(17)  INTEGER DATA (
0000H, 2000H, 3400H, 4C00H, /* A random function */
5D00H, 6A00H, 7200H, 7800H,
7B00H, 7D00H, 7600H, 6D00H,
5D00H, 4B00H, 3400H, 2200H,
1000H);

DMPY:    PROCEDURE (A,B) LONGINT EXTERNAL;
        DECLARE (A,B) INTEGER;
END DMPY;

LOOP:
TEMP=SHR(IN_VAL,4); /* TEMP is the most significant 4 bits of IN_VAL */
TABLE_LOW=TABLE(TEMP); /* If "TEMP" was replaced by "SHR(IN_VAL,4)" */
TABLE_HIGH=TABLE(TEMP+1); /* The code would work but the 8096 would */
/* do two shifts */

TABLE_DIF=TABLE_HIGH-TABLE_LOW;

OUT_DIF=DMPY(TABLE_DIF,SIGNED(IN_VAL AND 0FH)) /16;

OUT=SAR((TABLE_LOW+OUT_DIF),4); /* SAR performs an arithmetic right shift,
/* in this case 4 places are shifted */

IF CARRY=0 THEN RESULT=OUT; /* Using the hardware flags must be done */
ELSE RESULT=OUT+1; /* with care to ensure the flag is tested */
/* in the desired instruction sequence */
GOTO LOOP;

/* END OF PLM-96 CODE */

END;

```

Listing 3-5. 32-Bit Result Multiply Procedure For PLM-96

```

$TITLE('MULT.APT: 16*16 multiply procedure for PLM-96')

SP      EQU      18H:word
EXTRN   PLMREG  !long
CSEG

PUBLIC DMPY ; Multiply two integers and return a
             ; longint result in AX, DX registers

DMPY:    POP     PLMREG+4 ; Load return address
POP      PLMREG          ; Load one operand
MUL      PLMREG,[SP]+    ; Load second operand and increment SP
BR       [PLMREG+4]      ; Return to PLM code.

END

```



Using PLM, code requires less lines, is much faster to write, and easier to maintain, but may take slightly longer to run. For this example, the assembly code generated by the PLM-96 compiler takes 56.75 microseconds to run instead of 30.75 microseconds. If PLM-96 performed the 32-bit result multiply instead of using the ASM-96 routine the PLM code would take 41.5 microseconds to run. The actual code listings are shown in Appendix A.

## 3.2. USING THE I/O SECTION

### 3.2.1. Using the HSI unit

One of the most frequent uses of the HSI is to measure the time between events. This can be used for frequency determination in lab instruments, or speed/acceleration information when connected to pulse type encoders. The code in Listing 3-6 can be used to determine the high and low times of the signals on two lines. This code can be easily expanded to 4 lines and can also be modified to work as an interrupt routine.

Frequently it is also desired to keep track of the number of events which have occurred, as well as how often they are occurring. By using a software counter this feature can be added to the above code. This code depends on the software responding to the change in line state before the line changes again. If this cannot be guaranteed then it may be necessary to use 2 HSI lines for each incoming line. In this case one HSI line would look for falling edges while the other looks for rising edges. The code in Listing 3-7 includes both the counter feature and the edge detect feature.

The uses for this type of routine are almost endless. In instrumentation it can be used to determine frequency on input lines, or perhaps baud rate for a self adjusting serial port. Section 4.2 contains an example of making a software serial port using the HSI unit. Interfacing to some form of mechanically generated position information is a very frequent use of the HSI. The applications in this category include motor control, precise positioning (print heads, disk drives, etc.),

Listing 3-6. Measuring Pulses Using The HSI Unit

```

$TITLE('PULSE.APT: Measuring pulses using the HSI unit')
$INCLUDE(DEMO96.INC)

rseg    at 28H

HIGH_TIME: dsw 1
LOW_TIME:  dsw 1
PERIOD:    dsw 1
HI_EDGE:   dsw 1
LO_EDGE:   dsw 1

cseg    at 2080H

LD      SP, #100H
LDB     TOC0, #00000001B ; Enable HSI 0
LDB     HSI_MODE, #00001111B ; HSI 0 look for either edge

wait:   ADD     PERIOD, HIGH_TIME, LOW_TIME ; Add high and low times
JBS     IOS1, 6, contin ; If FIFO is full
JBC     IOS1, 7, wait ; Wait while no pulse is entered

contin: LDB     AL, HSI_STATUS ; Load status; Note that reading
; HSI_TIME clears HSI_STATUS

LD      BX, HSI_TIME ; Load the HSI TIME
JBS     AL, 1, hsi_hi ; Jump if HSI.0 is high

hsi_lo: ST      BX, LO_EDGE
SUB     HIGH_TIME, LO_EDGE, HI_EDGE
BR      wait

hsi_hi: ST      BX, HI_EDGE
SUB     LOW_TIME, HI_EDGE, LO_EDGE
BR      wait

END

```

engine control and transmission control. The HSI unit is used extensively in the example in section 4.3.

### 3.2.2. Using the HSO Unit

Although the HSO has many uses, the best example is that of a multiple PWM output. This program, shown in Listing 3-8, is simple enough to be easily understood, yet it shows how to use the HSO for a task which can be complex. In order for this program to operate, another program needs to set up the on and off time variables for each line. The program also requires that a HSO line not change so quickly that

it changes twice between consecutive reads of I/O Status Register 0, (IOS0).

A very eye catching example can be made by having the above program output waveforms that vary over time. The driver routine in Listing 3-10 can be linked to the above program to provide this function. Linking is accomplished using RL96, the relocatable linker for the 8096. Information for using RL96 can be found in the "MCS-96 Utilities Users Guide", listed in the bibliography. In order for the program to link, the register declaration section (ie. the section between

Listing 3-7. Enhanced HSI Pulse Measurement Routine

```

$TITLE ('ENHSI.APT: ENHANCED HSI PULSE ROUTINE')
$INCLUDE (DEMO96.INC)

RSEG AT 28H
TIME: DSW 1
LAST_RISE: DSW 1
LAST_FALL: DSW 1
HSI_S0: DSB 1
IOS1_BAK: DSB 1
PERIOD: DSW 1
LOW_TIME: DSW 1
HIGH_TIME: DSW 1
COUNT: DSW 1

cseg at 2080H
init: LD SP,#100H
LDB IOC1,#00100101B ; Disable HSO.4,HSO.5, HSI_INT=first,
; Enable PWM,TXD,TIMER1_OVRFLOW_INT

LDB HSI_MODE,#10011001B ; set hsi.1 -; hsi.0 +
LDB IOC0,#00000111B ; Enable hsi 0,1
; T2 CLOCK=T2CLK, T2RST=T2RST
; Clear timer2

wait: ANDB IOS1_BAK,#01111111B ; Clear IOS1_BAK.7
ORB IOS1_BAK,IOS1 ; Store into temp to avoid clearing
; other flags which may be needed
; If hsi is not triggered then
JBC IOS1_BAK,7,wait ; jump to wait

ANDB HSI_S0,HSI_STATUS,#01010101B
LD TIME,HSI_TIME
JBS HSI_S0,0,a_rise
JBS HSI_S0,2,a_fall
BR no_cnt

a_rise: SUB LOW_TIME,TIME, LAST_FALL
SUB PERIOD,TIME, LAST_RISE
LD LAST_RISE,TIME
BR increment

a_fall: SUB HIGH_TIME,TIME, LAST_RISE
SUB PERIOD,TIME, LAST_FALL
LD LAST_FALL,TIME

increment:
INC COUNT

no_cnt: BR wait

END

```

Listing 3-8. Generating a PWM with the HSO

```

$TITLE ('HSOPWM.APT: 8096 EXAMPLE PROGRAM FOR PWM OUTPUTS')
; This program will provide 3 PWM outputs on HSO pins 0-2
; The input parameters passed to the program are:
;
;           HSO_ON_N   HSO on time for pin N
;           HSO_OFF_N  HSO off time for pin N
;
;       Where: Times are in timer1 cycles
;              N takes values from 0 to 3
;
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
$INCLUDE (DEMO96.INC)
RSEG AT 28H

      HSO_ON_0:      DSW      1
      HSO_OFF_0:     DSW      1
      HSO_ON_1:      DSW      1
      HSO_OFF_1:     DSW      1
      OLD_STAT:      dsb      1
      NEW_STAT:      dsb      1

cseg   AT 2080H

      LD      SP, #100H
      LD      HSO_ON_0, #100H      ; Set initial values
      LD      HSO_OFF_0, #400H     ; Note that times must be long enough
      LD      HSO_ON_1, #280H     ; to allow the routine to run after each
      LD      HSO_OFF_1, #280H     ; line change.
      ANDB    OLD_STAT, IOS0, #0FH
      XORB    OLD_STAT, #0FH

wait:  JBS     IOS0, 6, wait        ; Loop until HSO holding register
      NOP                                     ; is empty

; For operation with interrupts 'store_stat:' would be the
; entry point of the routine.
; Note that a DI or PUSHF might have to be added.

store_stat:
      ANDB    NEW_STAT, IOS0, #0FH      ; Store new status of HSO
      CMPB    OLD_STAT, NEW_STAT
      JE      wait                    ; If status hasn't changed
      XORB    OLD_STAT, NEW_STAT

check_0:
      JBC     OLD_STAT, 0, check_1      ; Jump if OLD_STAT(0)=NEW_STAT(0)
      JBS     NEW_STAT, 0, set_off_0

set_on_0:
      LDB     HSO_COMMAND, #00110000B   ; Set HSO for timer1, set pin 0
      ADD     HSO_TIME, TIMER1, HSO_OFF_0 ; Time to set pin = Timer1 value
      BR      check_1                  ; + Time for pin to be low

set_off_0:
      LDB     HSO_COMMAND, #00010000B   ; Set HSO for timer1, clear pin 0
      ADD     HSO_TIME, TIMER1, HSO_ON_0 ; Time to clear pin = Timer1 value
      BR      check_1                  ; + Time for pin to be high

check_1:
      JBC     OLD_STAT, 1, check_done    ; Jump if OLD_STAT(1)=NEW_STAT(1)
      JBS     NEW_STAT, 1, set_off_1

set_on_1:
      LDB     HSO_COMMAND, #00110001B   ; Set HSO for timer1, set pin 1
      ADD     HSO_TIME, TIMER1, HSO_OFF_1 ; Time to set pin = Timer1 value
      BR      check_done

set_off_1:
      LDB     HSO_COMMAND, #00010001B   ; Set HSO for timer1, clear pin 1
      ADD     HSO_TIME, TIMER1, HSO_ON_1 ; Time to clear pin = Timer1 value
      BR      check_done

check_done:
      LDB     OLD_STAT, NEW_STAT        ; Store current status and
      ; wait for interrupt flag

      BR      wait

; use RET if "wait" is called from another routine

      END

```

"RSEG" and "CSEG") in Listing 3-8 must be changed to that in Listing 3-9.

The driver routine simply changes the duty cycle of the waveform and sets the second HSO output to a

frequency twice that of the first one. A slightly different driver routine could easily be the basis for a switching power supply or a variable frequency/variable voltage motor driver. The listing of the driver routine is shown in Listing 3-10.

### Listing 3-9. Changes to Declarations for HSO Routine

```

; NOTE: Use this file to replace the declaration section of
; the HSO PWM program from "$INCLUDE(DEMO96.INC)" through
; the line prior to the label "wait". Also change the last
; branch in the program to a "RET".
;
RSEG

D_STAT:      DSB      1
extrn  HSO_ON_0 :word , HSO_OFF_0 :word
extrn  HSO_ON_1 :word , HSO_OFF_1 :word
extrn  HSO_TIME :word , HSO_COMMAND :byte
extrn  TIMER1   :word , IOS0      :byte
extrn  SP       :word

public OLD_STAT
OLD_STAT:      dsb      1
NEW_STAT:      dsb      1

cseg
PUBLIC wait

```

### Listing 3-10. Driver Module for HSO PWM Program

```

$TITLE('HSODRV.APT: Driver module for HSO PWM program')

HSODRV      MODULE MAIN, STACKSIZE(8)

PUBLIC      HSO_ON_0 , HSO_OFF_0
PUBLIC      HSO_ON_1 , HSO_OFF_1
PUBLIC      HSO_TIME , HSO_COMMAND
PUBLIC      SP , TIMER1 , IOS0

$INCLUDE(DEMO96.INC)

rseg at 28H

EXTRN      OLD_STAT :byte
HSO_ON_0:   dsb      1
HSO_OFF_0:  dsb      1
HSO_ON_1:   dsb      1
HSO_OFF_1:  dsb      1
count:     dsb      1

cseg at 2080H

EXTRN      wait :entry

strt:      DI
LD         SP, #100H
ANDB      OLD_STAT, IOS0, #0FH
XORB      OLD_STAT, #0FH

initial:
LD         CX, #0100H

loop:      LD         AX, #1000H
SUB       BX, AX, CX
LD         AX, CX

ST         AX, HSO_ON_0
ST         BX, HSO_OFF_0

```



```

SHR     AX, #1
SHR     BX, #1
ST      AX, HSO_ON
ST      BX, HSO_OFF_1

CALL    wait

INC     CX
CMP     CX, #00F00H
BNE     loop
BR      initial
END

```

Since the 8096 needs to keep track of events which often repeat at set intervals it is convenient to be able to have Timer 2 act as a programmable modulo counter. There are several ways of doing this. The first is to program the HSO to reset Timer 2 when Timer 2 equals a set value. A software timer set to interrupt at Timer 2 equals zero could be used to reload the CAM. This software method takes up two locations in the CAM and does not synchronize Timer 2 to the external world.

To synchronize Timer 2 externally the T2 RST (Timer2 ReSeT) pin can be used. In this way Timer 2 will get reset on each rising edge of T2 RST. If it is desired to have an interrupt generated and time recorded when Timer 2 gets reset, the signal for its reset can be taken from HSI.0 instead of T2RST. The HSI.0 pin has its own interrupt vector which functions independently of the HSI unit.

Another option available is to use the HSI.1 pin to clock Timer2. By using this approach it is possible to use the HSI to measure the period of events on the input to Timer2. If both of the HSI pins are used instead of the T2RST and T2CLK pins the HSI0 unit can keep track of speed and position of the rotating device with very little software overhead. This type of setup is ideal for a system like the one shown in Figure 3-1, and similar to the one used in section 4.3.

### 3.2.3. Using the Serial Port in Mode 1

Mode 1 of the serial port supports the basic asynchronous 8-bit protocol and is used to interface to most CRTs and printers. The example in Listing 3-17 shows a simple routine which receives a character and then transmits the same character. The code is

Another option available is to use the HSI.1 pin to clock Timer2. By using this approach it is possible to use the HSI to measure the period of events on the input to Timer2. If both of the HSI pins are used instead of the T2RST and T2CLK pins the HSI0 unit can keep track of speed and position of the rotating device with very little software overhead. This type of setup is ideal for a system like the one shown in Figure 3-1, and similar to the one used in section 4.3.

In this system a sequence of events is required based on the position of the gear which represents any piece of rotating machinery. Timer 2 holds the count of the number of tooth edges passed since the index mark. By using HSI.1 as the input to Timer 2, instead of T2 CLK, it is possible to determine tooth count and time information through the HSI. From this information instantaneous velocity and acceleration can be calculated. Having the tooth edge count in Timer 2

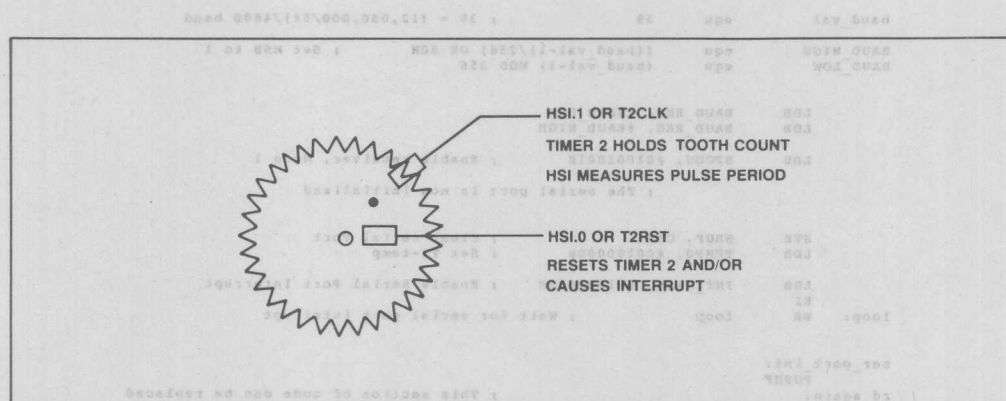


Figure 3-1. Using the HSI0 to Monitor Rotating Machinery

means that the HSO unit can be used to initiate the desired tasks at the appropriate tooth count. The interrupt routine initiated by HSI.0 can be used to perform any software task required every revolution. In this system, the overhead which would normally require extensive software has been done with the hardware on the 8096, thus making more software time available for control programs.

### 3.2.3. Using the Serial Port in Mode 1

Mode 1 of the serial port supports the basic asynchronous 8-bit protocol and is used to interface to most CRTs and printers. The example in Listing 3-11 shows a simple routine which receives a character and then transmits the same character. The code is

Listing 3-11. Using the Serial Port in Mode 1

```

;TITLE('SP.APT: SERIAL PORT DEMO PROGRAM')
$INCLUDE(Demo96.INC)
rseg at 28H
    CHR:   dsb 1
    SPTMP: dsb 1
    TEMP0: dsb 1
    TEMP1: dsb 1
    RCV_FLAG: dsb 1
cseg at 200CH
    DCW ser_port_int
cseg at 2080H
    LD SP, #100H
    LDB IOCL, #00100000B ; Set P2.0 to TXD
    ; Baud rate = input frequency / (64*baud_val)
    ; baud_val = (input frequency/64) / baud rate
    baud_val equ 39 ; 39 = (12,000,000/64)/4800 baud
    BAUD_HIGH equ ((baud_val-1)/256) OR 80H ; Set MSB to 1
    BAUD_LOW equ (baud_val-1) MOD 256
    LDB BAUD_REG, #BAUD_LOW
    LDB BAUD_REG, #BAUD_HIGH
    LDB SPCON, #01001001B ; Enable receiver, Mode 1
    ; The serial port is now initialized
    STB SBUF, CHR ; Clear serial Port
    LDB TEMP0, #00100000B ; Set TI-temp
    LDB INT_MASK, #01000000B ; Enable Serial Port Interrupt
    EI
loop: BR loop ; Wait for serial port interrupt

ser_port_int:
    PUSHF
rd_again:
    LDB SPTMP, SPSTAT ; This section of code can be replaced
    ORB TEMP0, SPTMP ; with "ORB TEMP0, SP_STAT" when the
    ANDB SPTMP, #01100000B ; serial port TI and RI bugs are fixed
    JNE rd_again ; Repeat until TI and RI are properly cleared

```

set up so that minor modifications could make it run on an interrupt basis. Note that it is necessary to set up some flags as initial conditions to get the routine to run properly. If it was desired to send 7 bits of data plus parity instead of 8 bits of data the PEN bit would be set to a one. Interprocessor communication, as described in section 2.3.4, can be set up by simply adding code to change RB8 and the port mode to the listing below. The hardware shown in Figure 3-2 can be used to convert the logic level output of the 8096 to  $\pm 12$  or 15 volt levels to connect to a CRT. This circuit has been found to work with most RS-232 devices, although it does not conform to strict RS-232 specifications. If true RS-232 conformance is required then any standard RS-232 driver can be used.



### 3.2.4. Using the A to D

The code in Listing 3-12 makes use of the software flags to implement a non-interrupt driven routine which scans A to D channels 0 through 3 and stores them as words in RAM. An interrupt driven routine is shown in section 4.1 When using the A to D it is important to always read the value using the byte read commands, and to give the converter 8 state times to start converting before reading the status bit.

Since there is no sample and hold on the A to D converter it may be desirable to use an RC filter on each input. A 100 ohm resistor in series with a 0.22 uf capacitor to ground has been used successfully in the lab. This circuit gives a time constant of around 22 microseconds which should be long enough to get rid of most noise, without overly slowing the A to D response time.

## 4.0 ADVANCED SOFTWARE EXAMPLES

Using the 8096 for applications which consist only of the brief examples in the previous section does not

really make use of its full capabilities. The following examples use some of the code blocks from the previous section to show how several I/O features can be used together to accomplish a practical task. Three examples will be shown. The first is simply a combination of several of the section 3 examples run under an interrupt system. Next, a software serial port using the HSIO unit is described. The concluding example is one of interfacing the HSI unit to an optical encoder to control a motor.

### 4.1. SIMULTANEOUS I/O ROUTINES UNDER INTERRUPT CONTROL

A four channel analog to PWM converter can easily be made using the 8096. In the example in Listing 4 analog channels are read and 3 PWM waveforms are generated on the HSO lines and one on the PWM pin. Each analog channel is used to set the duty cycle of its associated output pin. The interrupt system keeps the whole program humming, providing time for a background task which is simply a 32 bit software counter. To show which routines are executing and

Listing 3-12. Scanning the A to D Channels

```

$TITLE('ATOD.APT: SCANNING THE A TO D CHANNELS')
$INCLUDE(DEMO96.INC)

RSEG    at 28H
        BL EQU    BX:BYTE
        DL EQU    DX:BYTE

RESULT_TABLE:
RESULT_1:    dsw    1
RESULT_2:    dsw    1
RESULT_3:    dsw    1
RESULT_4:    dsw    1

cseg     at 2080H

start:    LD      SP, #100H      ; Set Stack Pointer
          CLR     BX

next:     ADDB     AD_COMMAND,BL, #1000B      ; Start conversion on channel
          ; indicated by BL register

          NOP      ; Wait for conversion to start
          NOP

check:    JBS     AD_RESULT_LO, 3, check      ; Wait while A to D is busy
          LDB      AL, AD_RESULT_LO           ; Load low order result
          LDB      AH, AD_RESULT_HI          ; Load high order result

          ADDB     DL, BL, BL                ; DL=BL*2
          LDBZ     DX, DL
          ST       AX, RESULT_TABLE[DX]      ; Store result indexed by BL*2

          INCB     BL                        ; Increment BL modulo 4
          ANDB     BL, #03H
          BR       next

END

```





### Listing 4-1b. Interrupt Driven HSO Routine

6-34

#### Listing 4-1c. Interrupt Driven A to D Routine

```

; A TO D COMPLETE INTERRUPT
;
ATOD_done int:
    PUSHF
    ORG Port1, #00000100B ; Set Pl.2

    ANDB AL, AD_RESULT_LO, #11000000B ; Load low order result
    LDB AH, AD_RESULT_HI ; Load high order result
    ADDB DL, AD_NUM, AD_NUM ; DL= AD_NUM *2
    LDBE DX, DL
    ST AX, RESULT_TABLE[DX] ; Store result indexed by DX

    CMPB AL, #01000000B
    JNH no_rnd ; Round up if needed
    CMPB AH, #0FFH ; Don't increment if AH=0FFH
    JE no_rnd
    INCB AH

no_rnd: LDB AL, AH ; Align byte and change to word
    CLRB AH
    ST AX, ON_TIME[DX]

    INCB AD_NUM
    ANDB AD_NUM, #03H ; Keep AD_NUM between 0 and 3

next: ADDB AD_COMMAND, AD_NUM, #1000B ; Start conversion on channel
    ; indicated by AD_NUM register
    ANDB Port1, #11111011B ; Clear Pl.2
    POPF
    RET
END

```

The HSO routine shown in Listing 4-1b is slightly different than the one in section 3. All of the HSO lines turn on at the same time, only the turn-off-time is varied between lines. This action is what is most commonly required for multiple PWM outputs and simplifies the software. A comparison is made between Timer1 and the next HSO turn on time at the beginning of the routine. If the next turn on time has passed, then the on-times are loaded into the CAM, otherwise the off times are loaded.

The maximum number of events in the CAM at any given time is 7. This occurs when the first line to turn off does so, causing the off-times for all of the lines to be loaded. For two of the lines there will be an off-time, an on-time, and the just loaded off-time. The other line (the one that just turned off) will have only the on-time and the just loaded off-time.

A/D conversions are performed by the code in Listing 4-1c about every 60 microseconds, 42 for the conversion, the rest for overhead. The A/D routine sets up the HSQ and PWM on and off times. Since the

A/D has a ten bit output, the most significant 8 bits are rounded up or down based on the least significant two bits.

## 4.2. SOFTWARE SERIAL PORT USING THE HSIO UNIT

There are many systems which require more than one serial port, an example is a system which must communicate with other computers and have an additional port for a local console. If the on-board UART is being used as an inter-processor link, the HSIO unit can be used to interface the 8096 to an additional asynchronous line.

Figure 4-1 shows the format of a standard 10-bit asynchronous frame. The start bit is used to synchronize the receiver to the transmitter; at the leading edge of the START bit the receiver must set up its timing logic to sample the incoming line in the center of each bit. Following the start bit are the eight data bits which are transmitted least significant bit first. The STOP bit is set to the opposite state of the START bit to

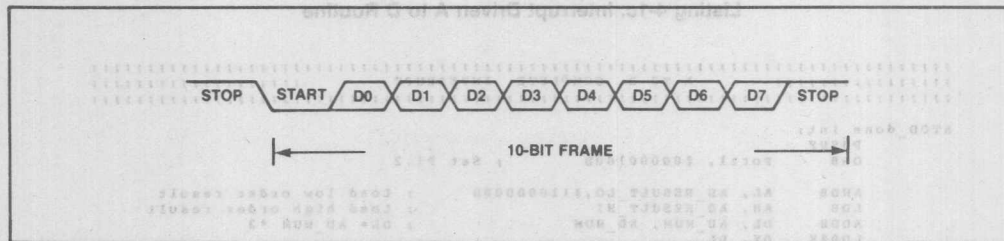


Figure 4-1. 10-bit Asynchronous Frame

guarantee that the leading edge of the START bit will cause a transition on the line; it also provides for a dead time on the line so that the receiver can maintain its synchronization.

The remainder of this section will show how a full-duplex asynchronous port can be built from the HSI0 unit. There are four sections to this code:

1. Interface routines. These routines provide a procedural interface between the interrupt driven core of the software serial port and the remainder of the application software.
2. Initialization routine. This routine is called during the initialization of the overall system and sets up the various variables used by the software port.

3. Transmit ISR. This routine runs as an ISR (interrupt service routine) in response to an HSO interrupt. Its function is to serialize the data passed to it by the interface routines.

4. Receive ISRs. There are two ISRs involved in the receive process. One of them runs in response to an HSI interrupt and is used to synchronize the receive process at the leading edge of the start bit. The second receive ISR runs in response to an HSO generated software timer interrupt, this routine is scheduled to run at the center of each bit and is used to deserialize the incoming data.

The routines share the set of variables that are shown in Listing 4-2. These variables should be accessed only by the routines which make up the software serial

Listing 4-2. Software Serial Port Declarations

```

;
; VARIABLES NEEDED BY THE SOFTWARE SERIAL PORT
;
;
; rseg
;
rcv_state: dsb 1
rxrdy equ 1 ; indicates receive done
rxoverrun equ 2 ; indicates receive overflow
rip equ 4 ; receive in progress flag
rcv_buf: dsb 1 ; used to double buffer receive data
rcv_reg: dsb 1 ; used to deserialize receive data
sample_time: dsb 1 ; records last receive sample time

serial_out: dsb 1 ; Holds the output character framing (start and
; stop bits) for transmit process.
baud_count: dsb 1 ; Holds the period of one bit in units
; of T1 ticks.
txd_time: dsb 1 ; Transition time of last Txdbit that was
; sent to the CAM
char: dsb 1 ; for test only
;
; COMMANDS ISSUED TO THE HSO UNIT
;
mark_command equ 0110101b ; timer1,set,interrupt on 5
space_command equ 0010101b ; timer1,clr,interrupt on 5
sample_command equ 0011000b ; software_timer 0
;
$seject

```



port. The table also shows the declarations for the commands issued to the HSO unit. In this example HSI.2 is used for receive data and HSO.5 is used for transmit data, although other HSI and HSO lines could have been used.

The interface routines are shown in Listing 4-3. Data is passed to the port by pushing the eight-bit character into the stack and calling *char\_out*, which waits for any in-process transmission to complete and stores the character into the variable *serial\_out*. As

the data is stored the START and STOP bits are added to the data bits. The routine *char\_in* is called when the application software requires a character from the port. The data is returned in the *ax* register in conformance to PLM 96 calling conventions. The routine *csts* can be called to determine if a character is available at the port before calling *char\_in*. ( If no character is available *char\_in* will wait indefinitely).

The initialization routine is shown in Listing 4-4. This routine is called with the required baud rate in the

Listing 4-3. Software Serial Port Interface Routines

```

; char_out:
; Output character to the software serial port
;
    pop     cx          ; the return address
    pop     bx          ; the character for output
    ldb     (bx+1),#01h ; add the start and stop bits
    add     bx,bx       ; to the char and leave as 16 bit
wait_for_xmit:
    cmp     serial_out,0 ; wait for serial_out=0 (it will be cleared by
    bne     wait_for_xmit ; the hso interrupt process)
    st      bx,serial_out ; put the formatted character in serial_out
    br      [cx]        ; return to caller

;
; csts:
; Returns "true" (ax<>0) if char_in has a character.
;
    clr     ax
    bbc     rcve_state,0,csts_exit
    inc     ax
csts_exit:
    ret

; char_in:
; Get a character from the software serial port
;
    ; wait for character ready
    bbc     rcve_state,0,char_in
    pushf   ; set up a critical region
    andb    rcve_state,#not(rxdy)
    ldbz    al,rcve_buf
    popf    ; leave the critical region
    ret

```

Listing 4-4. Software Serial Port Initialization Routine

```

;
; setup_serial_port:
; Called on system reset to initiate the software serial port.
;
    pop     cx          ; the return address
    pop     bx          ; the baud rate (in decimal)
    ld      dx,#00007h ; dx:ax:=500,000 (assumes 12 Mhz crystal)
    ld      ax,#0A120h ; calculate the baud count (500,000/baudrate)
    divu    ax,bx
    st      ax,baud_count
    st      0,serial_out ; clear serial_out
    ldb     ioc1,#01100000b ; Enable HSO.5 and Txd
    bbs     ioc0,6,$     ; Wait for room in the HSO CAM
    add     txd_time,timer1,20 ; and issue a MARK command.
    ldb     hso_command,#mark_command
    ld      hso_time,txd_time
    clrb    rcve_buf     ; clear out the receive variables
    clrb    rcve_reg
    clrb    rcve_state
    call    init_receive ; setup to detect a start bit
    br      [cx]        ; return

```

stack; it calculates the bit time from the baud rate and stores it in the variable *baud\_count* in units of TIMER1 ticks. An HSO command is issued which will initiate the transmit process and then the remainder of the variables owned by the port are initialized. The routine *init\_receive* is called to setup the HSI unit to look for the leading edge of the START bit.

The transmit process is shown in Listing 4-5. The HSO unit is used to generate an output command to the transmit pin once per bit time. If the *serial\_out* register is zero a MARK (idle condition) is output. If the *serial\_out* register contains data then the least

significant bit is output and the register shifted right one place. The framing information (START and STOP bits) are appended to the actual data by the interface routines. Note that this routine will be executed once per bit time whether or not data is being transmitted. It would be possible to use this routine for additional low resolution timing functions with minimal overhead.

The receive process consists of an initialization routine and two interrupt service routines, *hsi\_isr* and *software\_timer\_isr*. The listings of these routines are shown in Listings 4-6a, 4-6b, and 4-6c respectively.

Listing 4-5. Software Serial Port Transmit Process

```

; hso_isr:
; Fields the hso interrupts and performs the serialization of the data.
; Note: this routine would be incorporated into the hso service strategy for an
; actual system.

cseg      at 2006h
dcw      hso_isr      ; Set up vector

cseg
pushf
add      txd_time,baud_count
cmp      serial_out,0    ; if character is done send a mark
be       send_mark
shr      serial_out,#1    ; else send bit 0 of serial out and shift
bc       send_mark      ; serial_out left one place.

send_space:
ldb      hso_command,#space_command
ld       hso_time,txd_time
br       hso_isr_exit

send_mark:
ldb      hso_command,#mark_command
ld       hso_time,txd_time

hso_isr_exit:
popf
ret

$object

```

Listing 4-6. Receive Process

Listing 4-6a. Software Serial Port Receive Initialization

```

; init_receive:
; Called to prepare the serial input process to find the leading edge of
; a start bit.

ldb      ioc0,$00000000b      ; disconnect change detector
ldb      hsi_mode,$00100000b   ; negative edges on HSI.2

flush_fifo:
orb      ioc1_save,ioc1
bbc      ioc1_save,7,flush_fifo_done
ldb      al,hsi_status
ld       ax,hsi_time          ; trash the fifo entry
andb     ioc1_save,#not(80h)    ; clear bit 7.
br       flush_fifo

flush_fifo_done:
ldb      ioc0,$00010000b      ; connect HSI.2 to detector
ret

```

## Listing 4-6b. Software Serial Port Start Bit Detect

```

; hsi_isr:
; Fields interrupts from the HSI unit, used to detect the leading edge
; of the START bit.
; Note: this routine would be incorporated into the HSI strategy of an actual
; system.
;
; cseg at 2004h
; dcw hsi_isr ; setup the interrupt vector
;
; cseg
; pushf
; push ax
; ldb al, hsi_status
; ldb sample_time, hsi_time
; bbs al, 4, exit_hsi
; bbs al, 7, $ ; wait for room in HSO holding reg
; ld ax, baud_count ; send out sample command in 1/2
; shl ax, #1 ; bit time
; add sample_time, ax
; ldb hso_command, $sample_command
; st sample_time, hso_time
; ldb loc0, $00000000b ; disconnect hsi.2 from change detector
; exit_hsi:
; pop ax
; popf
; ret

```

## Listing 4-6c. Software Serial Port Data Reception

```

; software_timer_isr:
; Fields the software timer interrupt, used to deserialize the incoming data.
; Note: this routine would be incorporated into the software timer strategy
; in an actual system.
;
; cseg at 200ah
; dcw software_timer_isr ; setup vector
;
; cseg
; pushf
; orb ioal_save, ioal
; andb ioal_save, #not(01h) ; clear bit 0
; andb 0, rcve_state, #0fch ; All bits except rxrdy and overrun=0
; bne process_data
;
; process_start_bit:
; bbs hsi_status, 5, start_ok
; call init_receive
; br software_timer_exit
;
; start_ok:
; orb rcve_state, $rip ; set receive in progress flag
; br schedule_sample
;
; process_data:
; bbs rcve_state, 7, check_stopbit
; shrb rcve_reg, #1
; bbs hsi_status, 5, datazero
; orb rcve_reg, #80h ; set the new data bit
;
; datazero:
; addb rcve_state, #10h ; increment bit count
; br schedule_sample
;
; check_stopbit:
; bbs hsi_status, 5, $ ; DEBUG ONLY
; ldb rcve_buf, rcve_reg
; orb rcve_state, $rxrdy
; andb rcve_state, #03h ; Clear all but ready and overrun bits
; call init_receive
; br software_timer_exit
;
; schedule_sample:
; bbs ioal, 7, $ ; wait for holding reg empty
; ldb hso_command, $sample_command
; add sample_time, baud_count
; st sample_time, hso_time
;
; software_timer_exit:
; popf
; ret

```

tively. The start bit is detected by the *hsi\_isr* which schedules a software timer interrupt in one-half of a bit time. This first sample is used to verify that the START bit has not ended prematurely (a protection against a noisy line). The software timer service routine uses the variable *rcve\_state* to determine whether it should check for a valid START bit, deserialize data, or check for a valid STOP bit. When a complete character has been received it is moved to the receive buffer and *init\_receive* is called to set up the receive process for the next character. This routine is also called when an error (e.g. invalid START bit) is detected.

Appendix C contains the complete listing of the routines and the simple loop which was used to initialize them and verify their operation. The test was run for several hours at 9600 baud with no apparent malfunction of the port.

#### 4.3. INTERFACING AN OPTICAL ENCODER TO THE HSI UNIT

Optical encoders are among one of the more popular devices used to determine position of rotating equipment. These devices output two pulse trains with edges that occur from 2 to 4000 times a revolution.

Frequently there is a third line which generates one pulse per revolution for indexing purposes. Figure 4-2 shows a six line encoder and typical waveforms. As can be seen, the two waveforms provide the ability to determine both position and direction. Since a microcontroller can perform real time calculations it is possible to determine velocity and acceleration from the position and time information.

Interfacing to the encoder can be an interesting problem, as it requires connecting mechanically generated electrical signals to the HSI unit. The problems arise because it is difficult to obtain the exact nature of the signals under all conditions.

The equipment used in the lab was a Pittman 9400 series gearmotor with a 600 line optical encoder from Vernitech. The encoder has to be carefully attached to the shaft to minimize any runout or endplay. Fortunately, Pittman has started marketing their motors with ball bearings and optical encoders already installed. It is recommended that the encoder be mounted to the motor using the exact specifications of the encoder manufacturer and/or a good machine shop.

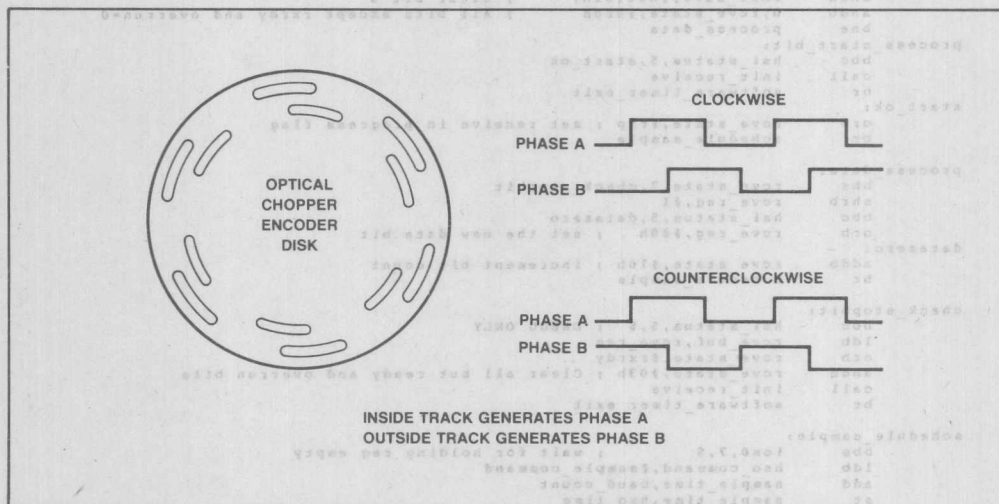


Figure 4-2. Optical Encoder and Waveforms



Digital filtering external to the 8096 is used on the encoder signals. The idealized signals coming from the encoder and after the digital filter are shown in Figure 4-3. The circuitry connecting the encoder to the 8096 requires only two chips. A one-shot constructed of XOR gates generates pulses on each edge of each signal. The pulses generated by Phase A are used to clock the signal from Phase B and vice versa. The hardware is shown in Figure 4-4. CMOS parts are used to reduce loading on the encoder so that buffers are not needed. Note that T2CLK is clocked on both edges of both filtered phases.

By using this method repetitive edges on a single phase without an edge on the other phase will not be passed on to the 8096. Repetitive edges on a phase can occur when the motor is stopped and vibrates or when it is changing direction. The digital filtering technique causes a little more delay in the signal at slow speeds than an analog filter would, but the simplicity trade off is worthwhile. The net effect of digital filtering is losing the ability to determine the first edge after a direction change. This does not affect the count since the first edge in both directions is lost.

If it is desired to determine when each edge occurs before filtering, the encoder outputs can be attached directly to the 8096. As these would be input signals, Port 0 is the most likely choice for connection. It would not be required to connect these lines to the HSI unit, as the information on them would only be needed when the motor is going very slowly.

The motor is driven using the PWM output pin for power control and a port pin for direction control. The 8096 drives a 7438 which drives 2 opto-isolators. These in turn drive two VFETs. A MOV (Metal Oxide Varistor, a type of transient absorber) is used to protect the VFETs, and a capacitor filters the PWM to get the best motor performance. Figure 4-5 shows the driver circuitry. To avoid noise getting into the 8096 system, the  $\pm 15$  volt power supply is isolated from the 8096 logic power supply.

This is the extent of the external circuitry required for this example. All of the counting and direction detection are done by the 8096. There are two sections to the example: driving the motor and interfacing to the encoder. The motor driver uses proportional control

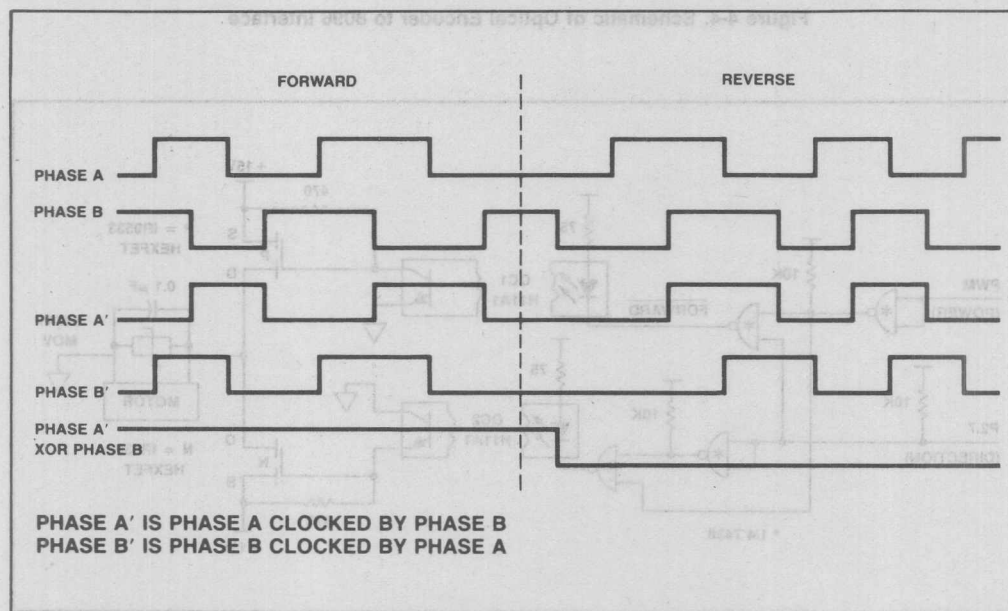


Figure 4-3. Filtered Encoder Waveforms

with some modifications and a braking algorithm. Since the main point of this example is I/O interfacing, the motor driver will be briefly described at the end of this section.

In order to interface to the encoder it is necessary to know the types of waveforms that can be expected. The motor was accelerated and decelerated many times using different maximum voltages. It was found

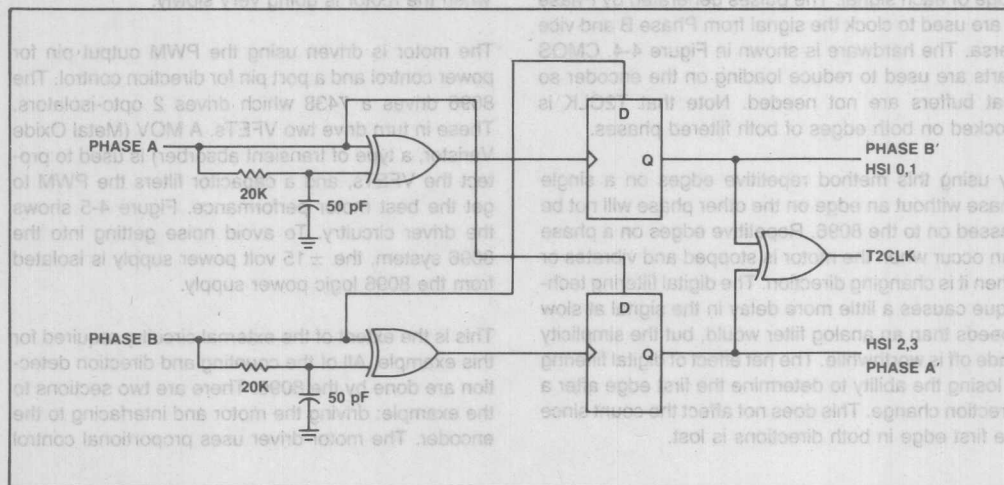


Figure 4-4. Schematic of Optical Encoder to 8096 Interface

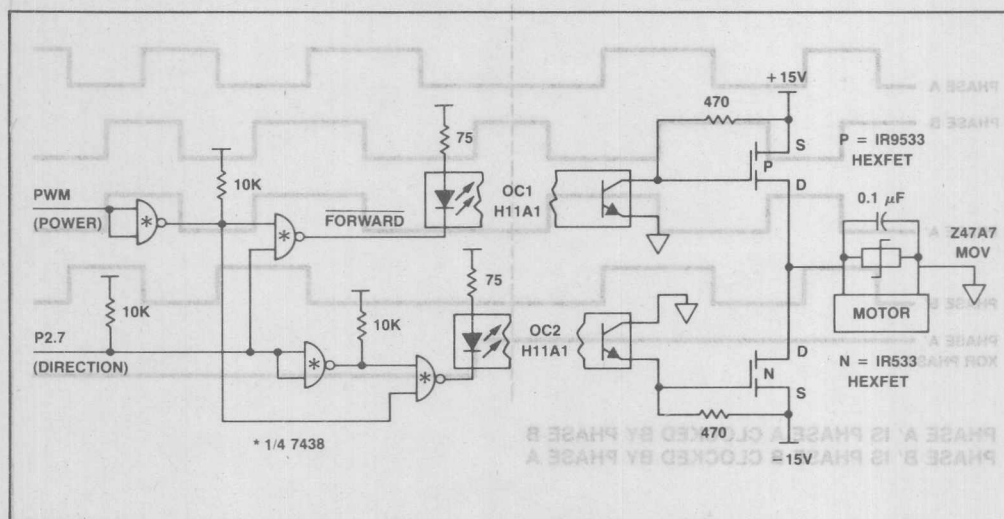


Figure 4-5. Motor Driver Circuitry

that the motor would decelerate smoothly until the time between encoder edges was around 100 microseconds. At this point the motor would either continue to decelerate slowly, or would suddenly stop and reverse. The latter case is the one that was most problematic.

After a brief overview, each section of the program will be described separately, with the complete listing included in the Appendix D. In order to make debugging easier, as well as to provide insight into how the program is working, I/O port 1 is used to indicate the program status. This information consists of which routine the program is in and under which mode it is operating. The main program sections are: Main loop, HSI interrupt, Timer 2 check, and Motor drive. There are also minor sections such as initialization, timer overflow handling, and software timer handling. Tying everything together is some overhead and glue. Where the glue is not obvious it will be discussed, otherwise it can be derived from the listings.

The program is a main loop which does nothing except serve as a place for the program to go when none of the interrupt routines are being run. All of the processing is done on an interrupt basis.

There are three basic software modes which are invoked depending on the speed of the motor. The modes referred to as 0, 1 and 2, in order from slowest to fastest operation. When the program is running the

operating mode is indicated by the lower 2 bits of Port 1, with the following coding:

| P1.0 | P1.1 | Mode | Description                     |
|------|------|------|---------------------------------|
| 0    | 0    | 0    | HSI looks at every edge         |
| 1    | 0    | 1    | HSI looks at Phase A edges only |
| 0    | 1    | 2    | Timer 2 used instead of HSI     |
| 1    | 1    | 2    | (Alternate form of above)       |

The example is easiest to see if mode 2 is described first, followed by mode 1 then mode 0. In mode 2 Timer 2 is used to count edges on the incoming signal. A software timer routine, which is actually run using HSO.0, uses the Timer 2 value to update a LONG (32-bit) software counter labeled *POSITION*. The HSO routine runs every 260 microseconds. The HSO.0 interrupt is used instead of an actual software timer because of the ability to easily unmask it while other software timer routines are running.

In the code in Listing 4-7, the mode is first determined. For the first pass ignore the code starting with the label *in\_mode\_1*. Starting with *in\_mode\_2* the counter is incremented or decremented based on bit zero of *DIRECT*. If *DIRECT.0*=0 the motor is going backward, if it is a 1 the motor is going forward. Next the count difference is checked to see if it is slow enough to go into mode 1. If not the routine returns to the code it was running when the interrupt occurred.

Listing 4-7. Motor Control HSO.0 Timer Routine

```

;=====
; SOFTWARE TIMER ROUTINE 0
; NOW USING HSO.0 TO TRIGGER
;=====
CSEG AT 2280H
hso_exec_int: ; Check mode - Update position in mode 2

    PUSHF
    ldb HSO_COMMAND, #30H
    add HSO_TIME, TIMER1, HSO0_dly

    orl port1, #00100000B
    ld Timer_2, TIMER2
    jbs port1, in_mode2

in_mode1:
    sub tmp1, Timer_2, old_t2
    cmp tmp1, #2
    jh end_swt0

; Check count difference in tmp1

set_mode0:
    jbc port1, 0, end_swt0
    andb port1, #11111100B
    ldb IOC0, #01010101B
    last_stat, zero
    end_swt0

; if already in mode 0
; Clear P1.0, P1.1 (set mode 0)
; enable all HSI mode

```

```

in_mode2:
    sub    delta_p,timer_2,tmr2_old    ; get timer2 count difference
    ld     tmr2_old,timer_2

    jbc     direct,0,in_rev

in_fwd:   add    position,delta_p
    addc    position+2,zero
    br      chk_mode

in_rev:   sub    position,delta_p
    subc    position+2,zero

chk_mode:
    sub     tmpl,timer_2,old_t2        ; Check count difference in tmpl
    cmp     tmpl,#5                    ; set model if count is too low
    jgt     end_sw0                     ; count <= 5

set_model:
    andb    Port1,#11111101B          ; Clear Pl.1, set Pl.0 (set mode 1)
    orb     Port1,#00000001B          ; enable HSI.0 and 1
    ld      IOC0,#00000101B
    ld      zero,HSI_TIME
    sub     last1_time,timer1,min_hsl1 ; set up so (time-last2_time)>min_hsl1 on next HSI
    clr     hsl:

    ld      ZERO,HSI_TIME
    andb    iosl_bak,#01111111B
    orb     iosl_bak,iosl
    jba     iosl_bak,7,clr_hsl        ; If hsl is triggered then clear hsl

end_sw0:
    ld      old_t2,TIMER_2
    andb    port1,#11011111B
    popf
    ret

```

If the pulse rate is slow enough to go to mode 1, the transition is made by enabling HSI.0 and HSI.1. Both of these lines are connected to the same encoder line, with HSI.0 looking for rising edges and HSI.1 looking for falling edges. The *HSI\_TIME* register is read to speed up clearing the HSI fifo and the *LAST1\_TIME* value is set up so the mode 1 routine does not immediately put the program into another mode. The HSI fifo is then cleared, the Timer2 value used throughout this routine is saved, and the routine returns.

This routine still runs in modes 0 and 1, but in an abbreviated form. The section of code starting with the label *in\_mode1* checks to see if the pulses are coming in so slowly that both HSI lines can be checked. If this is the case then all of the HSIs are enabled and the program returns. This routine is the secondary method for going from mode 1 to mode 0, the primary method is by checking the time between edges during the HSI routine, which will be described later.

The HSO routine will enable mode 0 from mode 1 if two edges are not received every 260 microseconds.

The primary method, (under the HSI routine), can only enable mode 0 after an edge is received. This could cause a problem if the last 2 edges on Phase A before the encoder stops were too close to enable mode 0. If this happened, mode 0 would not be enabled until after the encoder started again, resulting in missed edges on Phase B. Using the HSO routine to switch from mode 1 to mode 0 eliminates this problem.

Figure 4-6 shows a state diagram of how the mode switching is done. As can be seen, there are two sources for most of the mode decisions. This helps avoid problems such as the one mentioned above.

When either Mode 1 or Mode 0 is enabled the HSI interrupt routine performs the counting of edges, while the HSO routine only ensures that the correct mode is running. The routines for modes 0 and 1 share the same initialization and completion sections, with the main body of code being different.

The initialization routine is similar to many HSI routines. The flags are checked to ensure that the HSI fifo data is valid, and then the fifo is read. Next, the main body of code (for either mode 0 or mode 1) is



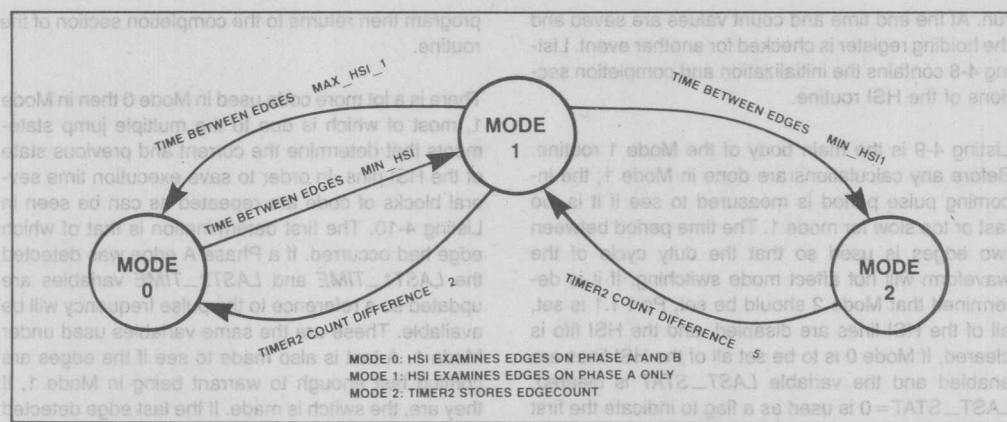


Figure 4-6. Mode State Diagram

## Listing 4-8. Motor Control HSI Data Available Routine

```

; HSI DATA AVAILABLE INTERRUPT ROUTINE
; This routine keeps track of the current time and position of the motor.
; The upper word of information is provided by the timer overflow routine.

CSEG AT 2400H
now_mode_1: br in_mode_1 ; used to save execution time for
no_intl: br no_int ; worst case loop

hsi_data_int: pushf
; set P1.6
orb portl,#01000000B
; Clear iosl_bak.7
andb iosl_bak,#01111111B
; If hsi is not triggered then
orb iosl_bak,iosl
jbc iosl_bak,7,no_intl ; jump to no_int

get_values:
ld timer_2,TIMER2
andb hsi_s0,HSI_STATUS,#01010101B
ld time,HSI_TIME

jbs portl,0,now_mode_1 ; jump if in mode 1

In_mode_0:

; INSERT BODY OF ROUTINE

load_last:
ld tmr2_old,timer_2
no_cnt: andb iosl_bak,#01111111B ; clr bit 7
orb iosl_bak,iosl
jbc iosl_bak,7,no_int
again: br get_values

no_int: andb portl,#01111111B ; Clear P1.6
popf
ret ; end of hsi_data interrupt routine
; Routine for mode 1 follows and then returns to "load_last"
$EJECT

```

run. At the end time and count values are saved and the holding register is checked for another event. Listing 4-8 contains the initialization and completion sections of the HSI routine.

Listing 4-9 is the main body of the Mode 1 routine. Before any calculations are done in Mode 1, the incoming pulse period is measured to see if it is too fast or too slow for mode 1. The time period between two edges is used so that the duty cycle of the waveform will not affect mode switching. If it is determined that Mode 2 should be set, Port 1.1 is set, all of the HSI lines are disabled, and the HSI fifo is cleared. If Mode 0 is to be set all of the HSI lines are enabled and the variable *LAST\_STAT* is cleared. *LAST\_STAT* = 0 is used as a flag to indicate the first HSI interrupt in Mode 0 after Mode 1. After the mode checking and setting are complete the incremental value in Timer 2 is used to update *POSITION*. The

program then returns to the completion section of the routine.

There is a lot more code used in Mode 0 than in Mode 1, most of which is due to the multiple jump statements that determine the current and previous state of the HSI pins. In order to save execution time several blocks of code are repeated as can be seen in Listing 4-10. The first determination is that of which edge had occurred. If a Phase A edge was detected the *LAST1\_TIME* and *LAST2\_TIME* variables are updated so a reference to the pulse frequency will be available. These are the same variables used under Mode 1. A test is also made to see if the edges are coming fast enough to warrant being in Mode 1, if they are, the switch is made. If the last edge detected was on Phase B, the information is used only to determine direction.

#### Listing 4-9. Motor Control Mode 1 Routines

```

In_mode_1:                ; mode 1 HSI routine
    andb    tmp1, hsi_s0, #01010000B
    jne     no_cnt
cmp_time:                ; Procedure which sets mode 1 also
    ld      last2_time, last1_time
    ld      last1_time, time

cmpl:    sub    tmp1, time, last2_time
    cmp      tmp1, min_hsl
    jh       check_max_time

set_mode_2:
    orb      Port1, #000000010B ; Set Pl.1 (in mode 2)
    ldb      IOC0, #00000000B ; Disable all HSI
mt_hsi:    ld      zero, hsi_time ; empty the hsi fifo
    andb     losl_bak, #01111111B ; clear bit 7
    orb      losl_bak, losl
    jbs      losl_bak, 7, mt_hsi ; If hsi is triggered then clear hsi
    br       done_chk

check_max_time:
    sub      tmp1, time, last2_time
    cmp      tmp1, max_hsl ; max_hsl = addition to min_hsl for
    jnh      done_chk ; total time

set_mode_0:
    andb     Port1, #11111100B ; clear Pl.0,1 set mode 0)
    ldb      IOC0, #01010101B ; Enable all HSI
    ldb      last_stat, zero

done_chk:
    sub      delta_p, timer_2, tmr2_old ; get timer2 count difference
    jbc      direct, 0, add_rev
add_fwd:    add      position, delta_p
    addc     position+2, zero
    br       load_last
add_rev:    sub      position, delta_p
    subc     position+2, zero
    br       load_last

$ject

```



After mode correctness is confirmed and the *LASTx\_TIME* values are updated the *LAST\_STAT* (Last Status) variable is used to determine the current direction of travel. The *POSITION* value is then updated in the direction specified by the last two edges and the status is stored. Note that the first time in Mode 0 after being in Mode 1, the Mode 1 *done\_chk* routine is used to update *POSITION*, instead of the routines *going\_fwd* and *going\_rev* from the Mode 0 section of code. The completion section of code is then executed.

Providing the PWM value to drive the motor is done by a routine running under Software Timer 1. The first section of code, shown in Listing 4-11a, has to do with calculating the position and time errors. Listing 4-11b shows the next section of code where the power to be supplied to the motor is calculated. First the direction is checked and if the direction is reverse the absolute value of the error is taken. If the error is greater than 64K counts, the PWM routine is loaded with the maximum value. The next check is made to

see if the motor is close enough to the desired location that the power to it should be reversed, (ie. enter the Braking mode). If the motor is very close to the position or has slowed to the point that is likely to turn around, the *Hold\_Position mode* is entered.

The determination of which modes are selected under what conditions was done empirically. All of the parameters used to determine the mode are kept in RAM so they can be easily changed on the fly instead of by re-assembling the program. The parameters in the listing have been selected to make the motor run, but have not been optimized for speed or stability. A diagram of the modes is shown in Figure 4-7.

In the *Hold\_Position* mode power is eased onto the motor to lock it into position. Since the motor could be stopped in this mode, some integral control is needed, as proportional control alone does not work well when the error is small and the load is large. The *BOOST* variable provides this integral control by increasing the output a fixed amount every time period

#### Listing 4-11. Motor Control Software Timer1 Routine

##### Listing 4-11a. Motor Control Software Position Counter

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; SOFTWARE TIMER ROUTINE 1 ;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

CSEG AT 2600H

swt1_expired:
    pushf
    orb     portl,%10000000B      ; set portl.7
    ldb     int_mask,%00001101B   ; enable HSI, Tofv, HSO
    ldb     HSO_COMMAND,%39H
    add     HSO_TIME,TIMER1,swt1_dly

    ld      time_err+2,des_time+2 ; Calculate time & position error
    ld      pos_err+2,des_pos+2
    sub     time_err,des_time,time ; values are set
    subc    time_err+2,time+2      ;
    sub     pos_err,des_pos,position
    subc    pos_err+2,position+2

EI

    sub     time_delta,last_time_err,time_err
    ld      last_time_err,time_err

    sub     pos_delta,last_pos_err,pos_err
    ld      last_pos_err,pos_err

    ; Time_err = Desired time to finish - current time
    ; Pos_err = Desired position to finish - current position
    ; Pos_delta = Last position error - Current position error
    ; Time_delta = Last time error - Current time error
    ; note that errors should get smaller so deltas will be
    ; positive for forward motion (time is always forward)

```



Listing 4-11b: Motor Control Power Algorithm

```

chk_dir:
    cmp     pos_err+2,zero
    jge     go_forward

go_backward:
    neg     pos_err          ; Pos_err = ABS VAL (pos_err)
    ldb     pwm_dir,#00h
    cmp     pos_err+2,#0ffffH
    jne     ld_max
    br      chk_brk

go_forward:
    ldb     pwm_dir,#01H
    cmp     pos_err+2,zero
    je      chk_brk

ld_max: ldb     pwm_pwr,max_pwr
        br      chk_sanity

chk_brk:
    cmp     pos_err,pos_pnt    ; Position_Error now = ABS(pos_err)
    jnh     hold_position      ; position_error < position_control_point
    cmp     pos_err,brk_pnt
    jh      ld_max             ; position_error > brake_point

braking:
    cmp     pos_delta,zero
    jge     chk_delta
    neg     pos_delta

chk_delta:
    cmp     pos_delta,vel_pnt    ; velocity = pos_delta/sample_time
    jnh     hold_position        ; jmp if ABS(velocity) < vel_pnt

brake: ldb     pwm_pwr,max_brk
        tmp,direct              ; If braking apply power in opposite
        notb tmp                ; direction of current motion
        ldb     pwm_dir,tmp

        br      ld_pwr

ld_pwr:
    cmp     pos_err,#02        ; position hold mode
    jh      calc_out           ; if position error < 2 then turn off power
    clr     tmp+2
    boost
    output

calc_out:
    mulub   tmp,max_hold,#255
    mulu    tmp,pos_err        ; Tmp = pos_err * max_hold
    cmp     pos_delta,zero
    jne     no_bst
    add     tmp+2,boost         ; Boost is integral control
                                ; TMP+2 = MSB(pos_err*max_hold)
    br      ck_max
no_bst: clr     boost
ck_max: cmp     tmp+2,max_hold
        jnh     output
maxed: ldb     tmp+2,max_hold
output: ldb     pwm_pwr,tmp+2

chk_sanity:
    br      ld_pwr

ld_pwr:
    ldb     rpwr,pwm_pwr
    notb    rpwr
    jbs     pwm_dir,0,p2fwd

p2bkwd: DI
        andb port2,#01111111B
        ldb     pwm_control,rpwr
        EI
        br      pwrset

p2fwd: DI
        orb    port2,#10000000B
        ldb     pwm_control,rpwr
        EI

```

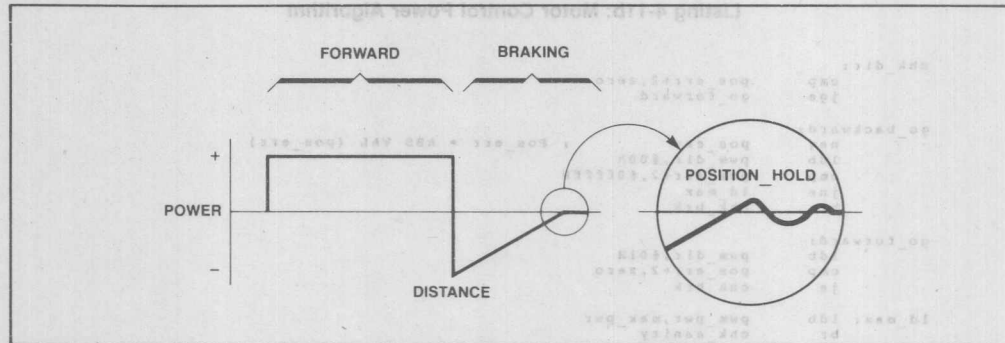


Figure 4-7. Motor Control Modes

in which the error does not get smaller. Once the error does get smaller, usually because the motor starts moving, BOOST is cleared.

A sanity check can be performed at this point to double check that the 8096 has proper control of the motor. In the example the worst that can happen is

Listing 4-12. Motor Control Next Position Lookup

```

pwrset:
    cmp     time_err+2,zero ; do pos_table when err is negative
    jgt     end_p
    br      end_p
; ; ;
    cmp     nxt_pos,#(32+pos_table)
    jlt     get_vals        ; jump if lower
    ld      nxt_pos,pos_table
    clr     time+2
get_vals:
    ld      des_pos,[nxt_pos]+
    ld      des_pos+2,[nxt_pos]+
    ld      des_time+2,[nxt_pos]+
    ld      max_pwr,[nxt_pos]+
    add     des_pos,offset
    addc    des_pos+2,zero
    sub     last_pos_err,des_pos,position
end_p:   andb    port1,#01111111B ; clear P1.7
    popf
    ret

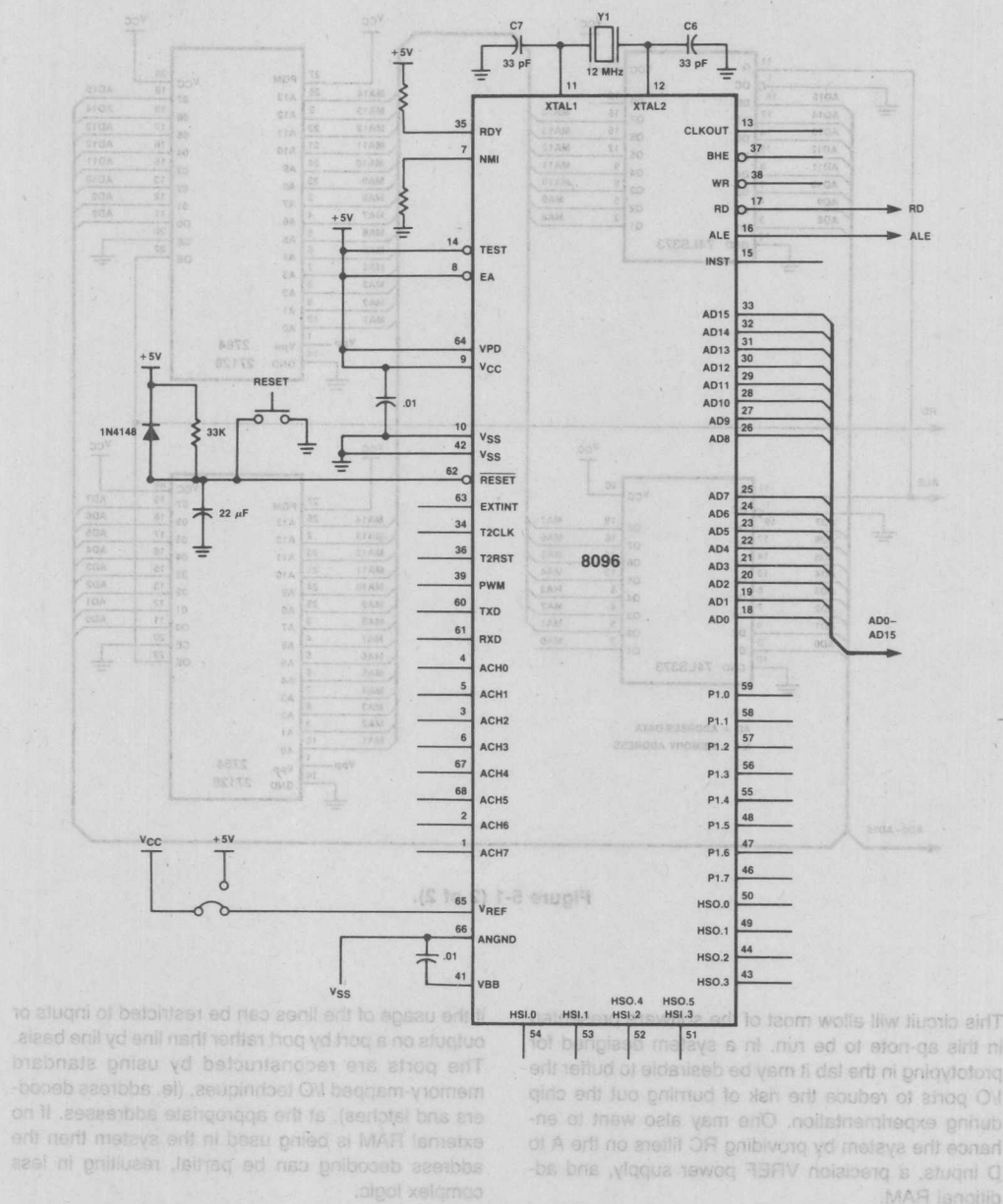
pos_table:
    dcl     00000000H ; position 0
    dcw     0020H, 0080H ; next time, power
    dcl     0000c000H ; position 1
    dcw     0040H, 0040H ; next time, power
    dcl     00000000H ; position 2
    dcw     0060H, 00c0H ; next time, power
    dcl     0FFFF800H ; position 3
    dcw     0080H, 0080H ; next time, power

    dcl     00000800H ; position 4
    dcw     0058H, 0080H ; next time, power
    dcl     00003000H ; position 5
    dcw     0070H, 00ffH ; next time, power
    dcl     00000000H ; position 6
    dcw     0090H, 00f0H ; next time, power
    dcl     00000000H ; position 7
    dcw     009H, 00f0H ; next time, power
    
```









the upper one contains the odd bytes, and the addressing is not fully decoded. This means that the addressing on a 2764 will be such that the lower 4K of each EPROM is mapped at 0000H and 4000H

while the upper 4K is mapped at 2000H. If the program being loaded is 16 Kbytes long the first half is loaded into the second half of the 2764s and vice versa. A similar situation exists when using 27128s.

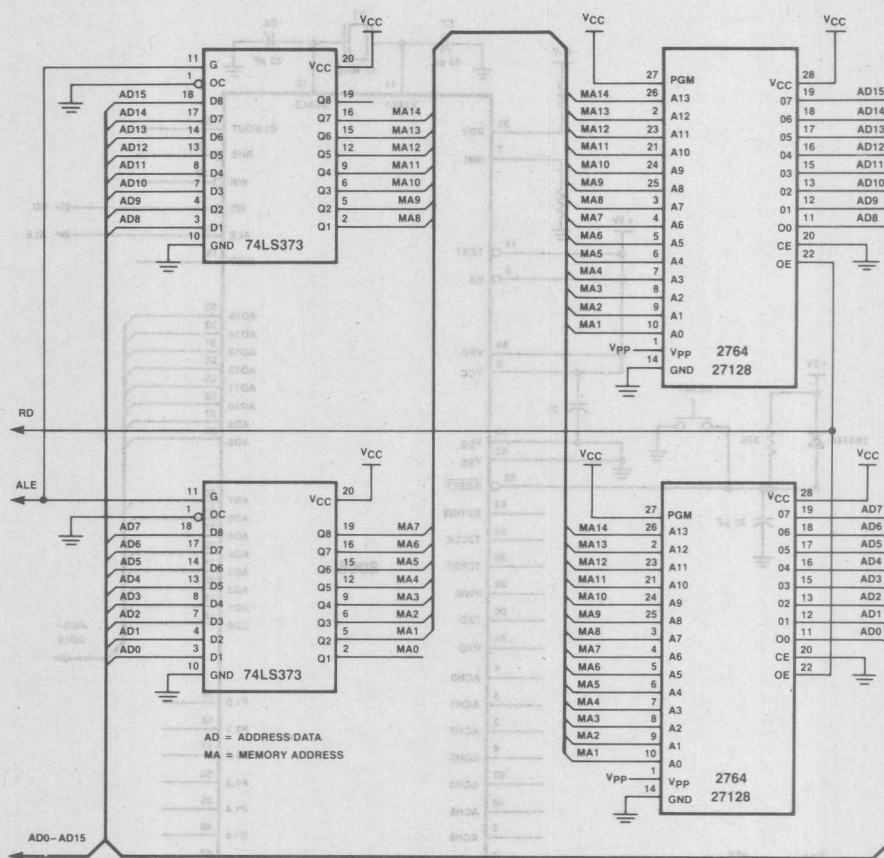


Figure 5-1 (2 of 2).

This circuit will allow most of the software presented in this ap-note to be run. In a system designed for prototyping in the lab it may be desirable to buffer the I/O ports to reduce the risk of burning out the chip during experimentation. One may also want to enhance the system by providing RC filters on the A to D inputs, a precision VREF power supply, and additional RAM.

## 5.2. PORT RECONSTRUCTION

If it is desired to fully emulate a 8396 then I/O ports 3 and 4 must be reconstructed. It is easiest to do this

if the usage of the lines can be restricted to inputs or outputs on a port by port rather than line by line basis. The ports are reconstructed by using standard memory-mapped I/O techniques, (ie. address decoders and latches), at the appropriate addresses. If no external RAM is being used in the system then the address decoding can be partial, resulting in less complex logic.

The reconstructed I/O ports will work with the same code as the on chip ports. The only difference will be the propagation delay in the external circuitry.

## 6.0 CONCLUSION

An overview of the MCS-96 family has been presented along with several simple examples and a few more complex ones. The source code for all of these programs are available in the Insite Users Library using order code AE-16. Additional information on the 8096 can be found in the Microcontroller Users Manual, and it is recommended that this book be in your possession before attempting any work with the MCS-96 family of products. Your local Intel sales office can assist you in getting more information on the 8096 and its hardware and software development tools.

## 7.0 BIBLIOGRAPHY

1. MCS-96 Users Manual (1984), Intel Corporation, 1983.  
Order number 230883-001
2. MCS-96 Macro Assembler User's Guide, Intel Corporation, 1983.  
Order number 122048-001
3. Microcontroller Handbook (1985), Intel Corporation, 1984.  
Order number 210918-002
4. MCS-96 Utilities User's Guide, Intel Corporation, 1983.  
Order number 122049-001
5. PL/M-96 User's Guide, Intel Corporation, 1983.  
Order number 122134-001







## APPENDIX A BASIC SOFTWARE EXAMPLES

### A.1. Table Lookup 1

SERIES-III MCS-96 MACRO ASSEMBLER, V1 0

SOURCE FILE: :F3:INTER1.A96

OBJECT FILE: :F3:INTER1.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

| ERR LOC | OBJECT          | LINE | SOURCE STATEMENT  |
|---------|-----------------|------|---|
|         |                 | 1    | \$TITLE('INTER1.A96: Interpolation routine 1')                |
|         |                 | 2    | !!!!!! 8096 Assembly code for table lookup and interpolation  |
|         |                 | 3    |   |
|         |                 | 4    | \$INCLUDE(:FO:DEMO96.INC) ; Include demo definitions          |
| =1      |                 | 5    | \$nolist ; Turn listing off for include file                  |
| =1      |                 | 53   | ; End of include file   |
|         |                 | 54   |   |
|         | 0022            | 55   | RSEG at 22H   |
|         |                 | 56   |   |
|         | 0022            | 57   | IN_VAL: dsb 1 ; Actual Input Value                            |
|         | 0024            | 58   | TABLE_LOW: dsb 1  |
|         | 0026            | 59   | TABLE_HIGH: dsb 1   |
|         | 0028            | 60   | IN_DIF: dsb 1 ; Upper Input - Lower Input                     |
|         | 0028            | 61   | IN_DIFB equ IN_DIF :byte                                      |
|         | 002A            | 62   | TAB_DIF: dsb 1 ; Upper Output - Lower Output                  |
|         | 002C            | 63   | OUT: dsb 1  |
|         | 002E            | 64   | RESULT: dsb 1   |
|         | 0030            | 65   | OUT_DIF: dsl 1 ; Delta Out                                    |
|         |                 | 66   |   |
|         |                 | 67   |   |
|         | 2080            | 68   | CSEG at 2080H   |
|         |                 | 69   |   |
|         | 2080 A1000118   | 70   | LD SP, #100H  |
|         |                 | 71   |   |
|         | 2084 B0221C     | 72   | look: LDB AL, IN_VAL ; Load temp with Actual Value            |
|         | 2087 18031C     | 73   | SHRB AL, #3 ; Divide the byte by 8                            |
|         | 208A 71FE1C     | 74   | ANDB AL, #11111110B ; Insure AL is a word address             |
|         |                 | 75   | ; This effectively divides AL by 2                            |
|         |                 | 76   | ; so AL = IN_VAL/16   |
|         |                 | 77   |   |
|         | 208D AC1C1C     | 78   | LDBZE AX, AL ; Load byte AL to word AX                        |
|         | 2090 A31D002124 | 79   | LD TABLE_LOW, TABLE [AX] ; TABLE_LOW is loaded with the value |
|         |                 | 80   | ; in the table at table location AX                           |
|         |                 | 81   |   |

```

2095 A31D022126      82      LD      TABLE_HIGH, (TABLE+2)[AX] ; TABLE_HIGH is loaded with the
83                      ; value in the table at table
84                      ; location AX+2
85      50A2 431D022126 86      LD      1VBGE TMC TMC 1VBGE TMC ; (The next value in the table)
87      209A 4824262A 88      SUB      TAB_DIF, TABLE_HIGH, TABLE_LOW ; TAB_DIF=TABLE_HIGH-TABLE_LOW
89      50A5 431D022126 90      ANDBE   IN_DIFB, IN_VAL, #0FH ; IN_DIFB=least significant 4 bits
91                      ; of IN_VAL
92      20A2 AC282B      93      LDBZE   IN_DIF, IN_DIFB ; Load byte IN_DIFB to word IN_DIF
93      50A5 431D022126 94      MULB   OUT_DIF, IN_DIF, TAB_DIF ; Output_difference =
94      50A5 431D022126 95      ; Input_difference*Table_difference
95      20AA 0E0430 1B 96      SHRAL   OUT_DIF, #4 ; Divide by 16 (2**4)
96      50A5 431D022126 97      ;
97      50A5 431D022126 98      ;
98      50A5 431D022126 99      ADD     OUT, OUT_DIF, TABLE_LOW ; Add output difference to output
99      50A5 431D022126 100     ; generated with truncated IN_VAL
100     50A5 431D022126 101     ; as input
101     50A5 431D022126 102     SHRA    OUT, #4 ; Round to 12-bit answer
102     50A5 431D022126 103     ADDC    OUT, zero ; Round up if Carry = 1
103     50A5 431D022126 104     ;
104     50A5 431D022126 105     no_inc: ST     OUT, RESULT ; Store OUT to RESULT
105     50A5 431D022126 106     ;
106     50A5 431D022126 107     BR     look ; Branch to "look."
107     50A5 431D022126 108     ;
108     50A5 431D022126 109     ;
109     50A5 431D022126 110     cseg   AT 2100H
110     50A5 431D022126 111     ;
111     50A5 431D022126 112     table: DCW   0000H, 2000H, 3400H, 4C00H ; A random function
112     50A5 431D022126 113     DCW   5D00H, 6A00H, 7200H, 7800H
113     50A5 431D022126 114     DCW   7B00H, 7D00H, 7600H, 6D00H
114     50A5 431D022126 115     DCW   5D00H, 4B00H, 3400H, 2200H
115     50A5 431D022126 116     DCW   1000H
116     50A5 431D022126 117     ;
117     50A5 431D022126 118     END

```

ASSEMBLY COMPLETED, NO ERROR(S) FOUND.

END LOC 0B7EC1 FILE SOURCE BIN/INTEL

COMMENTS SPECIFIED IN INVOCATION COMMAND: NONE

OBJECT LIFE: 43 IN/OUTS 0B7

SOURCE LIFE: 43 IN/OUTS 0B7

SERIES-III MCS-86 MICRO ASSEMBLER AT 0

AS 13016 looknb 5

## A.2. Table Lookup 2

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F3:INTER2.A96

OBJECT FILE: :F3:INTER2.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

| ERR LOC | OBJECT | LINE | SOURCE STATEMENT   |
|---------|--------|------|--|
|         |        | 1    | \$TITLE('INTER2.A96: Interpolation routine 2')                   |
|         |        | 2    |  |
|         |        | 3    | 8096 Assembly code for table lookup and interpolation            |
|         |        | 4    | Using tabled values in place of division                         |
|         |        | 5    |  |
|         |        | 6    | \$INCLUDE(:F0:DEMO96.INC) ; Include demo definitions             |
|         |        | 7    | \$nolist ; Turn listing off for include file                     |
|         |        | 8    | End of include file  |
|         |        | 56   |  |
|         |        | 57   | RSEG at 24H  |
|         |        | 58   |  |
|         |        | 59   | IN_VAL: dsb 1 ; Actual Input Value                               |
|         |        | 60   | TABLE_LOW: dsb 1 ; Table value for function                      |
|         |        | 61   | TABLE_INC: dsb 1 ; Incremental change in function                |
|         |        | 62   | IN_DIF: dsb 1 ; Upper Input - Lower Input                        |
|         |        | 63   | IN_DIFB equ IN_DIF : byte  |
|         |        | 64   | OUT: dsb 1   |
|         |        | 65   | RESULT: dsb 1  |
|         |        | 66   | OUT_DIF: dsb 1 ; Delta Out                                       |
|         |        | 67   |  |
|         |        | 68   |  |
|         |        | 69   | CSEG at 2080H  |
|         |        | 70   |  |
|         |        | 71   | LD SP, #100H ; Initialize SP to top of reg. file                 |
|         |        | 72   |  |
|         |        | 73   | look: LDB AL, IN_VAL ; Load temp with Actual Value               |
|         |        | 74   | SHRB AL, #3 ; Divide the byte by 8                               |
|         |        | 75   | ANDB AL, #1111110B ; Insure AL is a word address                 |
|         |        | 76   | DBIE IN_DIFB ; This effectively divides AL by 2                  |
|         |        | 77   | so AL = IN_VAL/16  |
|         |        | 78   | LDBZE AX, AL ; Load byte AL to word AX                           |
|         |        | 79   |  |
|         |        | 80   | LD TABLE_LOW, VAL_TABLE[AX] ; TABLE_LOW is loaded with the value |
|         |        | 81   | in the value table at location AX                                |
|         |        | 82   |  |
|         |        | 83   | LD TABLE_INC, INC_TABLE[AX] ; TABLE_INC is loaded with the value |
|         |        | 84   | in the increment table at  |
|         |        | 85   | location AX+2  |
|         |        | 86   |  |



```

209A 510F242A      87          ANDB     IN_DIFB, IN_VAL, #0FH      ; IN_DIFB=least significant 4 bits
209E AC2A2A        88          ; of IN_VAL
209E AC2A2A        89      LDBZ     IN_DIF, IN_DIFB      ; Load byte IN_DIFB to word IN_DIF
20A1 FE4C2B2A30    91      DBL     OUT_DIF, IN_DIF, TABLE_INC ;
20A1 FE4C2B2A30    92          ; Output_difference =
20A1 FE4C2B2A30    93      DBL     OUT_DIF, IN_DIF, TABLE_INC ; Input_difference*Incremental_change
20A1 FE4C2B2A30    94          ;
20A6 4426302C      95      ADD     OUT, OUT_DIF, TABLE_LOW ; Add output difference to output
20A6 4426302C      96      DBL     OUT, OUT_DIF, TABLE_LOW ; generated with truncated IN_VAL
20A6 4426302C      97      DBL     OUT, OUT_DIF, TABLE_LOW ; as input
20AA 0B042C        98      SHR     OUT, #4              ; Round to 12-bit answer
20AD A4002C        99      DBL     OUT, OUT_DIF, TABLE_INC ; Round up if Carry = 1
20B0 C02E2C       100          ;
20B0 C02E2C       101      no_inc: ST     OUT, RESULT      ; Store OUT to RESULT
20B3 27CF          102      DBL     BR     look          ; Branch to "look:"
20B3 27CF          103          ;
20B3 27CF          104      DBL     BR     look          ; Branch to "look:"
2100              105      cseg     AT 2100H
2100              106      DBL     BR     look          ; Branch to "look:"
2100              107      val_table: DBL     BR     look          ; Branch to "look:"
2100 000000200034004C 108      DBL     BR     look          ; Branch to "look:"
2108 005D006A0072007B 109      DBL     BR     look          ; Branch to "look:"
2110 007B007D0076006D 110      DBL     BR     look          ; Branch to "look:"
2118 005D004B00340022 111      DBL     BR     look          ; Branch to "look:"
2120 0010          112      DBL     BR     look          ; Branch to "look:"
2122              113      val_inc_table: DBL     BR     look          ; Branch to "look:"
2122 0002400180011001 114      DBL     BR     look          ; Branch to "look:"
212A D000800060003000 115      DBL     BR     look          ; Branch to "look:"
2132 200090FF70FF00FF 116      DBL     BR     look          ; Branch to "look:"
213A E0FE90FEE0FEE0FE 117      DBL     BR     look          ; Branch to "look:"
2142              118      DBL     BR     look          ; Branch to "look:"
2142              119      DBL     BR     look          ; Branch to "look:"

```

ASSEMBLY COMPLETED, NO ERROR(S) FOUND.

### A.3. PLM-96 Code with Expansion

SERIES-III PL/M-96 V1.0 COMPILATION OF MODULE PLMEX  
OBJECT MODULE PLACED IN :F3:PLMEX1.OBJ  
COMPILER INVOKED BY: PLM96.B6 :F3:PLMEX1.P96 CODE

\$TITLE('PLMEX1: PLM-96 Example Code for Table Lookup')

/\* PLM-96 CODE FOR TABLE LOOK-UP AND INTERPOLATION \*/

```

1      PLMEX: DO;
2      1      DECLARE IN_VAL      WORD      PUBLIC;
3      1      DECLARE TABLE_LOW  INTEGER   PUBLIC;
4      1      DECLARE TABLE_HIGH INTEGER   PUBLIC;
5      1      DECLARE TABLE_DIF  INTEGER   PUBLIC;
6      1      DECLARE OUT_DCH      INTEGER   PUBLIC;
7      1      DECLARE RESULT_DCH  INTEGER   PUBLIC;
8      1      DECLARE OUT_DIFCH    LONGINT   PUBLIC;
9      1      DECLARE TEMP_DIF    WORD      PUBLIC;
10     1      DECLARE TABLE(17)  INTEGER   DATA;
11     1      DMPY: PROCEDURE (A,B) LONGINT EXTERNAL;
12     2      DECLARE (A,B) INTEGER;
13     2      END DMPY;
14     1      LOOP:
15     1      TEMP=SHR(IN_VAL,4); /* TEMP is the most significant 4 bits of IN_VAL */
16     1      TABLE_LOW=TABLE(TEMP); /* If "TEMP" was replaced by "SHR(IN_VAL,4)" */
17     1      TABLE_HIGH=TABLE(TEMP+1); /* The code would work but the 8096 would */
18     1      /* do two shifts */
19     1      TABLE_DIF=TABLE_HIGH-TABLE_LOW;
20     1      OUT_DIF=DMPY(TABLE_DIF,SIGNED(IN_VAL AND OFH)) /16;
21     1      OUT=SAR((TABLE_LOW+OUT_DIF),4); /* SAR performs an arithmetic right shift,
22     1      in this case 4 places are shifted */

```

```

20 1      IF CARRY=0 THEN RESULT=OUT; /* Using the hardware flags must be done */
22 1      ELSE RESULT=OUT+1;          /* with care to ensure the flag is tested */
                                         /* in the desired instruction sequence */
23 1      GOTO LOOP;

                                         /* END OF PLM-96 CODE */

24 1      END;

```

PL/M-96 COMPILER PLMEX1: PLM-96 Example Code for Table Lookup  
ASSEMBLY LISTING OF OBJECT CODE

```

; STATEMENT 14
PLMEX:
0022 A100001B R LD SP, #STACK
0026 LOOP:
0026 A00010 R LD TEMP, IN_VAL
0029 0B0410 R SHR TEMP, #4H
; STATEMENT 15
002C 4410101C R ADD TMP0, TEMP, TEMP
0030 A31D000002 R LD TABLE_LOW, TABLE[TMP0]
; STATEMENT 16
0035 A31D020004 R LD TABLE_HIGH, TABLE+2H[TMP0]
; STATEMENT 17
003A 4B020406 R SUB TABLE_DIF, TABLE_HIGH, TABLE_LOW
; STATEMENT 18
003E C806 R PUSH TABLE_DIF
0040 410F00001C R AND TMP0, IN_VAL, #0FH
0045 C81C R PUSH TMP0
0047 EF0000 E CALL DMPY
004A 0E041C R SHRAL TMP0, #4H
004D A01E0E R LD OUT_DIF+2H, TMP2
0050 A01C0C R LD OUT_DIF, TMP0
; STATEMENT 19
0053 A00220 R LD TMP4, TABLE_LOW
0056 0620 50005: EXT TMP4
005B 641C20 ADD TMP4, TMP0
005B A41E22 ADDC TMP6, TMP2
005E 0E0420 SHRAL TMP4, #4H
0061 A0200B R LD OUT, TMP4
; STATEMENT 20
0064 B1FF1C LDB TMP0, #0FFH
0067 DB02 BC @0003
0069 111C CLR B TMP0
006B @0003: @0003:
006B A81C00

```

```

006B 981C00      CMPB  RO, TMP0
006E D705        BNE  @0001
                ; STATEMENT 21
0070 A0200A      LD    RESULT, TMP4
0073 2005        BR    @0002
                ; STATEMENT 22
                @0001:
0075 A0080A      LD    RESULT, OUT
0078 070A        INC   RESULT
                ; STATEMENT 23
                @0002:
007A 27AA        BR    LOOP
                ; STATEMENT 24
                END
0080 V01C0C      ;
008D V01E0E      ;
009V 0E0V1C      ;
MODULE INFORMATION:
                E
CODE AREA SIZE      = 005AH  90D
CONSTANT AREA SIZE  = 0022H  34D
DATA AREA SIZE      = 0000H  0D
STATIC REGS AREA SIZE = 0012H  18D
0039 V31B050004  ;
0030 V31B000005  ;
003C V31B010101B ;
PL/M-96 COMPILER  PLMEX1.  PLM-96 Example Code for Table Lookup
ASSEMBLY LISTING OF OBJECT CODE
005A 080-        ;
005F V00C10      ;
0078            ;
OVERLAYABLE REGS. AREA SIZE = 0000H  0D
MAXIMUM STACK SIZE         = 0006H  6D
48 LINES READ
PL/M-96 COMPILATION COMPLETE.      0 WARNINGS,      0 ERRORS
VERBODEN TO KOPIEEREN OF OBJECT CODE
PL/M-96 COMPILER  PLMEX1.  PLM-96 Example Code for Table Lookup
54  ;      END
      ;* END OF PLM-96 CODE *
53  ;      0010 100B
55  ;      BTST RESULT, OUT+1
50  ;      IF CARRY=0 THEN RESULT=OUT
      ;* TO THE NEXT INSTRUCTIONS
      ;* WITH CODE 00 000000 0000 0000
      ;* 0000 0000 0000 0000
      ;* 0000 0000 0000 0000

```



MCS-96 MACRO ASSEMBLER MULT.APT: 16\*16 multiply procedure for PLM-96

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F3:MULT.A96

OBJECT FILE: :F3:MULT.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

| ERR LOC       | OBJECT | LINE | SOURCE STATEMENT   |
|---------------|--------|------|--|
|               |        | 1    | \$TITLE('MULT.APT: 16*16 multiply procedure for PLM-96') |
|               |        | 2    |  |
|               |        | 3    |  |
| 0018          |        | 4    | SP EQU 18H:word  |
|               |        | 5    |  |
| 0000          |        | 6    | rseg   |
|               |        | 7    | EXTRN PLMREG :long                                       |
|               |        | 8    |  |
| 0000          |        | 9    | cseg   |
|               |        | 10   |  |
|               |        | 11   | PUBLIC DMPY ; Multiply two integers and return a         |
|               |        | 12   | ; longint result in AX, DX registers                     |
|               |        | 13   |  |
| 0000 CC04     | E      | 14   | DMPY: POP PLMREG+4 ; Load return address                 |
| 0002 CC00     | E      | 15   | POP PLMREG ; Load one operand                            |
| 0004 FE6E1900 | E      | 16   | MUL PLMREG,[SP]+ ; Load second operand and increment SP  |
|               |        | 17   |  |
| 0008 E304     | E      | 18   | BR [PLMREG+4] ; Return to PLM code.                      |
| 000A          |        | 19   | END  |

ASSEMBLY COMPLETED, NO ERROR(S) FOUND.

-----

SECTION1 NAME LOC L3:PLM001.DAT(PLMEX)

PLM001.FIB(PLMEX) 11A05183  
L3:PLM001.DAT(PLMEX) 11A05183  
L3:PLM001.DAT(PLMEX) 11A05183  
INPL:MOD001.FIB(PLMEX)

MOD(0000H-0000H)  
CONTROLS SPECIFIED IN INVOCATION COMMAND  
OBJFILE L3:PLM001.DAT  
INPL:MOD001.FIB(PLMEX) L3:PLM001.DAT(PLMEX) L3:PLM001.DAT(PLMEX)

COMPLT: 1683 INPL: 00000000  
SERIES-III MCS-96 REGULATOR AND PLMEX V1.0

intel®

MCS-96

1683

6-05

SERIES-III MCS-96 RELOCATOR AND LINKER, V2.0  
Copyright 1983 Intel Corporation

INPUT FILES: :F3:PLMEX1.OBJ, :F3:MULT.OBJ, PLM96.LIB  
OUTPUT FILE: :F3:PLMOUT.OBJ  
CONTROLS SPECIFIED IN INVOCATION COMMAND:  
ROM(2080H-3FFFFH)

INPUT MODULES INCLUDED:  
:F3:PLMEX1.OBJ(PLMEX) 12/25/84  
:F3:MULT.OBJ(MULT) 12/25/84  
PLM96.LIB(PLMREG) 11/02/83

SEGMENT MAP FOR :F3:PLMOUT.OBJ(PLMEX):

|             | TYPE | BASE  | LENGTH | ALIGNMENT | MODULE NAME |
|-------------|------|-------|--------|-----------|-------------|
|             | ---- | ----  | -----  | -----     | -----       |
| **RESERVED* |      | 0000H | 001AH  |           |             |
| *** GAP *** |      | 001AH | 0002H  |           |             |
| REG         |      | 001CH | 0008H  | ABSOLUTE  | PLMREG      |
| REG         |      | 0024H | 0012H  | WORD      | PLMEX       |
| STACK       |      | 0036H | 0006H  | WORD      |             |
| *** GAP *** |      | 003CH | 2044H  |           |             |
| CODE        |      | 2080H | 0003H  | ABSOLUTE  | PLMEX       |
| *** GAP *** |      | 2083H | 0001H  |           |             |
| CODE        |      | 2084H | 007CH  | WORD      | PLMEX       |
| CODE        |      | 2100H | 000AH  | BYTE      | MULT        |
| *** GAP *** |      | 210AH | DEF6H  |           |             |

SYMBOL TABLE FOR : F3: PLMOUT. OBJ (PLMEX)

| ATTRIBUTES       | VALUE | NAME         |
|------------------|-------|--------------|
| REG 5040 WORD    | 0024H | IN_VAL       |
| REG 5041 INTEGER | 0026H | TABLE_LOW    |
| REG 5042 INTEGER | 0028H | TABLE_HIGH   |
| REG 5043 INTEGER | 002AH | TABLE_DIF    |
| REG 5044 INTEGER | 002CH | OUT          |
| REG 5045 INTEGER | 002EH | RESULT       |
| REG 5046 LONGINT | 0030H | OUT_DIF      |
| REG 5047 WORD    | 0034H | TEMP         |
| CODE 5048 ENTRY  | 2100H | DMPY         |
| REG 5049 LONGINT | 001CH | PLMREG       |
| NULL 5050 NULL   | 003CH | MEMORY       |
| NULL 5051 NULL   | 1FC4H | ?MEMORY_SIZE |

5051 810E03  
5052 810112  
5053 410C011B

NAME

-----

5054 810E03

5055 810112

5056 410C011B

5057 810E03

5058 810112

5059 410C011B

5060 810E03

5061 810112

5062 410C011B

5063 810E03

5064 810112

5065 410C011B

5066 810E03

5067 810112

5068 410C011B

5069 810E03

5070 810112

5071 410C011B

5072 810E03

5073 810112

5074 410C011B

5075 810E03

5076 810112

5077 410C011B

5078 810E03

5079 810112

5080 410C011B

5081 810E03

5082 810112

5083 410C011B

5084 810E03

5085 810112

5086 410C011B

5087 810E03

5088 810112

5089 410C011B

5090 810E03

5091 810112

5092 410C011B

5093 810E03

5094 810112

5095 410C011B

5096 810E03

5097 810112

5098 410C011B

5099 810E03

5100 810112

5101 410C011B

5102 810E03

5103 810112

5104 410C011B

5105 810E03

5106 810112

5107 410C011B

5108 810E03

5109 810112

5110 410C011B

5111 810E03

5112 810112

5113 410C011B

5114 810E03

5115 810112

5116 410C011B

5117 810E03

5118 810112

5119 410C011B

5120 810E03

5121 810112

5122 410C011B

5123 810E03

5124 810112

5125 410C011B

5126 810E03

5127 810112

5128 410C011B

5129 810E03

5130 810112

5131 410C011B

5132 810E03

5133 810112

5134 410C011B

5135 810E03

5136 810112

5137 410C011B

5138 810E03

5139 810112

5140 410C011B

5141 810E03

5142 810112

5143 410C011B

5144 810E03

5145 810112

5146 410C011B

5147 810E03

5148 810112

5149 410C011B

5150 810E03

5151 810112

5152 410C011B

5153 810E03

5154 810112

5155 410C011B

5156 810E03

5157 810112

5158 410C011B

5159 810E03

5160 810112

5161 410C011B

5162 810E03

5163 810112

5164 410C011B

5165 810E03

5166 810112

5167 410C011B

5168 810E03

5169 810112

5170 410C011B

5171 810E03

5172 810112

5173 410C011B

5174 810E03

5175 810112

5176 410C011B

5177 810E03

5178 810112

5179 410C011B

5180 810E03

5181 810112

5182 410C011B

5183 810E03

5184 810112

5185 410C011B

5186 810E03

5187 810112

5188 410C011B

5189 810E03

5190 810112

5191 410C011B

5192 810E03

5193 810112

5194 410C011B

5195 810E03

5196 810112

5197 410C011B

5198 810E03

5199 810112

5200 410C011B

5201 810E03

5202 810112

5203 410C011B

5204 810E03

5205 810112

5206 410C011B

5207 810E03

5208 810112

5209 410C011B

5210 810E03

5211 810112

5212 410C011B

5213 810E03

5214 810112

5215 410C011B

5216 810E03

5217 810112

5218 410C011B

5219 810E03

5220 810112

5221 410C011B

5222 810E03

5223 810112

5224 410C011B

5225 810E03

5226 810112

5227 410C011B

5228 810E03

5229 810112

5230 410C011B

5231 810E03

5232 810112

5233 410C011B

5234 810E03

5235 810112

5236 410C011B

5237 810E03

5238 810112

5239 410C011B

5240 810E03

5241 810112

5242 410C011B

5243 810E03

5244 810112

5245 410C011B

5246 810E03

5247 810112

5248 410C011B

5249 810E03

5250 810112

5251 410C011B

5252 810E03

5253 810112

5254 410C011B

5255 810E03

5256 810112

5257 410C011B

5258 810E03

5259 810112

5260 410C011B

5261 810E03

5262 810112

5263 410C011B

5264 810E03

5265 810112

5266 410C011B

5267 810E03

5268 810112

5269 410C011B

5270 810E03

5271 810112

5272 410C011B

5273 810E03

5274 810112

5275 410C011B

5276 810E03

5277 810112

5278 410C011B

5279 810E03

5280 810112

5281 410C011B

5282 810E03

5283 810112

5284 410C011B

5285 810

## A.4. Pulse Measurement

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F3:PULSE.A96

OBJECT FILE: :F3:PULSE.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

| ERR | LOC           | OBJECT | LINE | SOURCE STATEMENT  |
|-----|---------------|--------|------|---|
|     |               |        | 1    | \$TITLE('PULSE.A96: Measuring pulses using the HSI unit')   |
|     |               |        | 2    |   |
|     |               |        | 3    | \$INCLUDE(DEMO96.INC)                                       |
| =1  |               |        | 4    | \$nolist ; Turn listing off for include file                |
| =1  |               |        | 52   | ; End of include file                                       |
|     |               |        | 53   |   |
|     | 0028          |        | 54   | rseg at 28H   |
|     |               |        | 55   |   |
|     | 0028          |        | 56   | HIGH_TIME: dsw 1  |
|     | 002A          |        | 57   | LOW_TIME: dsw 1   |
|     | 002C          |        | 58   | PERIOD: dsw 1   |
|     | 002E          |        | 59   | HI_EDGE: dsw 1  |
|     | 0030          |        | 60   | LO_EDGE: dsw 1  |
|     |               |        | 61   |   |
|     |               |        | 62   |   |
|     |               |        | 63   |   |
|     | 2080          |        | 64   | cseg at 2080H   |
|     |               |        | 65   |   |
|     |               |        | 66   |   |
|     | 2080 A100011B |        | 67   | LD SP, #100H  |
|     | 2084 B10115   |        | 68   | LDB IOCO, #00000001B ; Enable HSI 0                         |
|     | 2087 B10F03   |        | 69   | LDB HSI_MODE, #00001111B ; HSI 0 look for either edge       |
|     |               |        | 70   |   |
|     | 208A 442A282C |        | 71   | wait: ADD PERIOD, HIGH_TIME, LOW_TIME                       |
|     | 208E 3E1603   |        | 72   | IOS1, 6, contin ; If FIFO is full                           |
|     | 2091 3716F6   |        | 73   | JBC IOS1, 7, wait ; Wait while no pulse is entered          |
|     |               |        | 74   |   |
|     | 2094 80061C   |        | 75   | contin: LDB AL, HSI_STATUS ; Load status; Note that reading |
|     |               |        | 76   | ; HSI_TIME clears HSI_STATUS                                |
|     |               |        | 77   |   |
|     | 2097 A00420   |        | 78   | LD BX, HSI_TIME ; Load the HSI_TIME                         |
|     |               |        | 79   |   |
|     | 209A 391C09   |        | 80   | JBS AL, 1, hsi_hi ; Jump if HSI.0 is high                   |
|     |               |        | 81   |   |
|     | 209D C03020   |        | 82   | hsi_lo: ST BX, LO_EDGE                                      |
|     | 20A0 482E302B |        | 83   | SUB HIGH_TIME, LO_EDGE, HI_EDGE                             |
|     | 20A4 27E4     |        | 84   | BR wait   |
|     |               |        | 85   |   |
|     |               |        | 86   |   |
|     | 20A6 C02E20   |        | 87   | hsi_hi: ST BX, HI_EDGE                                      |



6-69

**MCS®-96**

© 2001

## A.5. Enhanced Pulse Measurement

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: F3:ENHSI.A96

OBJECT FILE: F3:ENHSI.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

| ERR LOC | OBJECT        | LINE | SOURCE STATEMENT   |
|---------|---------------|------|--|
|         |               | 1    | \$TITLE ('ENHSI.A96: ENHANCED HSI PULSE ROUTINE')            |
|         |               | 2    |  |
|         |               | 3    | \$INCLUDE(DEMO96.INC)  |
| =1      |               | 4    | \$nolist ; Turn listing off for include file                 |
| =1      |               | 52   | ; End of include file  |
|         |               | 53   |  |
|         | 0028          | 54   | RSEG AT 28H  |
|         |               | 55   |  |
|         | 0028          | 56   | TIME: DSW 1  |
|         | 002A          | 57   | LAST_RISE: DSW 1   |
|         | 002C          | 58   | LAST_FALL: DSW 1   |
|         | 002E          | 59   | HSI_S0: DSB 1  |
|         | 002F          | 60   | IOS1_BAK: DSB 1  |
|         | 0030          | 61   | PERIOD: DSW 1  |
|         | 0032          | 62   | LOW_TIME: DSW 1  |
|         | 0034          | 63   | HIGH_TIME: DSW 1   |
|         | 0036          | 64   | COUNT: DSW 1   |
|         |               | 65   |  |
|         | 2080          | 66   | cseg at 2080H  |
|         |               | 67   |  |
|         | 2080 A100011B | 68   | init: LD SP,#100H  |
|         |               | 69   |  |
|         | 2084 B12516   | 70   | LDB IOC1,#00100101B ; Disable HSD. 4, HSD. 5, HSI_INT=first, |
|         |               | 71   | ; Enable PWM, TXD, TIMER1_OVRFLOW_INT                        |
|         |               | 72   |  |
|         | 2087 B19903   | 73   | LDB HSI_MODE,#10011001B ; set hsi.1 -, hsi.0 +               |
|         | 208A B10715   | 74   | LDB IOC0,#00000111B ; Enable hsi.0,1                         |
|         |               | 75   | ; T2 CLOCK=T2CLK, T2RST=T2RST                                |
|         |               | 76   | ; Clear timer2   |
|         |               | 77   |  |
|         |               | 78   |  |
|         | 208D 717F2F   | 79   | wait: ANDB IOS1_BAK,#01111111B ; Clear IOS1_BAK.7            |
|         | 2090 90162F   | 80   | ORB IOS1_BAK,IOS1 ; Store into temp to avoid clearing        |
|         |               | 81   | ; other flags which may be needed                            |
|         | 2093 372FF7   | 82   | JBC IOS1_BAK,7,wait ; If hsi is not triggered then           |
|         |               | 83   | ; jump to wait   |
|         |               | 84   |  |
|         | 2096 5155062E | 85   | ANDB HSI_S0,HSI_STATUS,#01010101B                            |
|         | 209A A00428   | 86   | LD TIME,HSI_TIME   |
|         | 209D 31B8     | 87   | DB #912  |
|         | 209E 19305E3V | 88   | 209 FOR TIME' HI'EDGE' TO'EDGE'                              |

```

209D 382E05      88      JBS      HSI_SO,0,a_rise
20A0 3A2E0F      89      JBS      HSI_SO,2,a_fall
20A3 201A        90      BR       no_cnt
20A4 4851C50     91      SUB      BY_VX,CX
20A5 482C2832    92      a_rise: SUB      LOW_TIME, TIME, LAST_FALL
20A9 482A2830    93      SUB      PERIOD, TIME, LAST_RISE
20AD A02B2A      94      LD       LAST_RISE, TIME
20B0 200B        95      INC      increment
20B2 482A2834    96      a_fall: SUB      HIGH_TIME, TIME, LAST_RISE
20B6 482C2830    97      SUB      PERIOD, TIME, LAST_FALL
20BA A02B2C      98      LD       LAST_FALL, TIME
20BD 200B        99      INC      increment
20BD 0736        100     INC      COUNT
20BF 27CC        101     BR       wait
20C1            102     END
20C3            103
20C5            104
20C7            105
20C9            106
20CB            107
20CD            108
20CF            109
20D1            110
20D3            111
20D5            112
20D7            113
20D9            114
20DB            115
20DD            116
20DF            117
20E1            118
20E3            119
20E5            120
20E7            121
20E9            122
20EB            123
20ED            124
20EF            125
20F1            126
20F3            127
20F5            128
20F7            129
20F9            130
20FB            131
20FD            132
20FF            133
2101            134
2103            135
2105            136
2107            137
2109            138
210B            139
210D            140
210F            141
2111            142
2113            143
2115            144
2117            145
2119            146
211B            147
211D            148
211F            149
2121            150
2123            151
2125            152
2127            153
2129            154
212B            155
212D            156
212F            157
2131            158
2133            159
2135            160
2137            161
2139            162
213B            163
213D            164
213F            165
2141            166
2143            167
2145            168
2147            169
2149            170
214B            171
214D            172
214F            173
2151            174
2153            175
2155            176
2157            177
2159            178
215B            179
215D            180
215F            181
2161            182
2163            183
2165            184
2167            185
2169            186
216B            187
216D            188
216F            189
2171            190
2173            191
2175            192
2177            193
2179            194
217B            195
217D            196
217F            197
2181            198
2183            199
2185            200
2187            201
2189            202
218B            203
218D            204
218F            205
2191            206
2193            207
2195            208
2197            209
2199            210
219B            211
219D            212
219F            213
21A1            214
21A3            215
21A5            216
21A7            217
21A9            218
21AB            219
21AD            220
21AF            221
21B1            222
21B3            223
21B5            224
21B7            225
21B9            226
21BB            227
21BD            228
21BF            229
21C1            230
21C3            231
21C5            232
21C7            233
21C9            234
21CB            235
21CD            236
21CF            237
21D1            238
21D3            239
21D5            240
21D7            241
21D9            242
21DB            243
21DD            244
21DF            245
21E1            246
21E3            247
21E5            248
21E7            249
21E9            250
21EB            251
21ED            252
21EF            253
21F1            254
21F3            255
21F5            256
21F7            257
21F9            258
21FB            259
21FD            260
21FF            261
2201            262
2203            263
2205            264
2207            265
2209            266
220B            267
220D            268
220F            269
2211            270
2213            271
2215            272
2217            273
2219            274
221B            275
221D            276
221F            277
2221            278
2223            279
2225            280
2227            281
2229            282
222B            283
222D            284
222F            285
2231            286
2233            287
2235            288
2237            289
2239            290
223B            291
223D            292
223F            293
2241            294
2243            295
2245            296
2247            297
2249            298
224B            299
224D            300
224F            301
2251            302
2253            303
2255            304
2257            305
2259            306
225B            307
225D            308
225F            309
2261            310
2263            311
2265            312
2267            313
2269            314
226B            315
226D            316
226F            317
2271            318
2273            319
2275            320
2277            321
2279            322
227B            323
227D            324
227F            325
2281            326
2283            327
2285            328
2287            329
2289            330
228B            331
228D            332
228F            333
2291            334
2293            335
2295            336
2297            337
2299            338
229B            339
229D            340
229F            341
22A1            342
22A3            343
22A5            344
22A7            345
22A9            346
22AB            347
22AD            348
22AF            349
22B1            350
22B3            351
22B5            352
22B7            353
22B9            354
22BB            355
22BD            356
22BF            357
22C1            358
22C3            359
22C5            360
22C7            361
22C9            362
22CB            363
22CD            364
22CF            365
22D1            366
22D3            367
22D5            368
22D7            369
22D9            370
22DB            371
22DD            372
22DF            373
22E1            374
22E3            375
22E5            376
22E7            377
22E9            378
22EB            379
22ED            380
22EF            381
22F1            382
22F3            383
22F5            384
22F7            385
22F9            386
22FB            387
22FD            388
22FF            389
2301            390
2303            391
2305            392
2307            393
2309            394
230B            395
230D            396
230F            397
2311            398
2313            399
2315            400
2317            401
2319            402
231B            403
231D            404
231F            405
2321            406
2323            407
2325            408
2327            409
2329            410
232B            411
232D            412
232F            413
2331            414
2333            415
2335            416
2337            417
2339            418
233B            419
233D            420
233F            421
2341            422
2343            423
2345            424
2347            425
2349            426
234B            427
234D            428
234F            429
2351            430
2353            431
2355            432
2357            433
2359            434
235B            435
235D            436
235F            437
2361            438
2363            439
2365            440
2367            441
2369            442
236B            443
236D            444
236F            445
2371            446
2373            447
2375            448
2377            449
2379            450
237B            451
237D            452
237F            453
2381            454
2383            455
2385            456
2387            457
2389            458
238B            459
238D            460
238F            461
2391            462
2393            463
2395            464
2397            465
2399            466
239B            467
239D            468
239F            469
23A1            470
23A3            471
23A5            472
23A7            473
23A9            474
23AB            475
23AD            476
23AF            477
23B1            478
23B3            479
23B5            480
23B7            481
23B9            482
23BB            483
23BD            484
23BF            485
23C1            486
23C3            487
23C5            488
23C7            489
23C9            490
23CB            491
23CD            492
23CF            493
23D1            494
23D3            495
23D5            496
23D7            497
23D9            498
23DB            499
23DD            500
23DF            501
23E1            502
23E3            503
23E5            504
23E7            505
23E9            506
23EB            507
23ED            508
23EF            509
23F1            510
23F3            511
23F5            512
23F7            513
23F9            514
23FB            515
23FD            516
23FF            517
2401            518
2403            519
2405            520
2407            521
2409            522
240B            523
240D            524
240F            525
2411            526
2413            527
2415            528
2417            529
2419            530
241B            531
241D            532
241F            533
2421            534
2423            535
2425            536
2427            537
2429            538
242B            539
242D            540
242F            541
2431            542
2433            543
2435            544
2437            545
2439            546
243B            547
243D            548
243F            549
2441            550
2443            551
2445            552
2447            553
2449            554
244B            555
244D            556
244F            557
2451            558
2453            559
2455            560
2457            561
2459            562
245B            563
245D            564
245F            565
2461            566
2463            567
2465            568
2467            569
2469            570
246B            571
246D            572
246F            573
2471            574
2473            575
2475            576
2477            577
2479            578
247B            579
247D            580
247F            581
2481            582
2483            583
2485            584
2487            585
2489            586
248B            587
248D            588
248F            589
2491            590
2493            591
2495            592
2497            593
2499            594
249B            595
249D            596
249F            597
24A1            598
24A3            599
24A5            600
24A7            601
24A9            602
24AB            603
24AD            604
24AF            605
24B1            606
24B3            607
24B5            608
24B7            609
24B9            610
24BB            611
24BD            612
24BF            613
24C1            614
24C3            615
24C5            616
24C7            617
24C9            618
24CB            619
24CD            620
24CF            621
24D1            622
24D3            623
24D5            624
24D7            625
24D9            626
24DB            627
24DD            628
24DF            629
24E1            630
24E3            631
24E5            632
24E7            633
24E9            634
24EB            635
24ED            636
24EF            637
24F1            638
24F3            639
24F5            640
24F7            641
24F9            642
24FB            643
24FD            644
24FF            645
2501            646
2503            647
2505            648
2507            649
2509            650
250B            651
250D            652
250F            653
2511            654
2513            655
2515            656
2517            657
2519            658
251B            659
251D            660
251F            661
2521            662
2523            663
2525            664
2527            665
2529            666
252B            667
252D            668
252F            669
2531            670
2533            671
2535            672
2537            673
2539            674
253B            675
253D            676
253F            677
2541            678
2543            679
2545            680
2547            681
2549            682
254B            683
254D            684
254F            685
2551            686
2553            687
2555            688
2557            689
2559            690
255B            691
255D            692
255F            693
2561            694
2563            695
2565            696
2567            697
2569            698
256B            699
256D            700
256F            701
2571            702
2573            703
2575            704
2577            705
2579            706
257B            707
257D            708
257F            709
2581            710
2583            711
2585            712
2587            713
2589            714
258B            715
258D            716
258F            717
2591            718
2593            719
2595            720
2597            721
2599            722
259B            723
259D            724
259F            725
25A1            726
25A3            727
25A5            728
25A7            729
25A9            730
25AB            731
25AD            732
25AF            733
25B1            734
25B3            735
25B5            736
25B7            737
25B9            738
25BB            739
25BD            740
25BF            741
25C1            742
25C3            743
25C5            744
25C7            745
25C9            746
25CB            747
25CD            748
25CF            749
25D1            750
25D3            751
25D5            752
25D7            753
25D9            754
25DB            755
25DD            756
25DF            757
25E1            758
25E3            759
25E5            760
25E7            761
25E9            762
25EB            763
25ED            764
25EF            765
25F1            766
25F3            767
25F5            768
25F7            769
25F9            770
25FB            771
25FD            772
25FF            773
2601            774
2603            775
2605            776
2607            777
2609            778
260B            779
260D            780
260F            781
2611            782
2613            783
2615            784
2617            785
2619            786
261B            787
261D            788
261F            789
2621            790
2623            791
2625            792
2627            793
2629            794
262B            795
262D            796
262F            797
2631            798
2633            799
2635            800
2637            801
2639            802
263B            803
263D            804
263F            805
2641            806
2643            807
2645            808
2647            809
2649            810
264B            811
264D            812
264F            813
2651            814
2653            815
2655            816
2657            817
2659            818
265B            819
265D            820
265F            821
2661            822
2663            823
2665            824
2667            825
2669            826
266B            827
266D            828
266F            829
2671            830
2673            831
2675            832
2677            833
2679            834
267B            835
267D            836
267F            837
2681            838
2683            839
2685            840
2687            841
2689            842
268B            843
268D            844
268F            845
2691            846
2693            847
2695            848
2697            849
2699            850
269B            851
269D            852
269F            853
26A1            854
26A3            855
26A5            856
26A7            857
26A9            858
26AB            859
26AD            860
26AF            861
26B1            862
26B3            863
26B5            864
26B7            865
26B9            866
26BB            867
26BD            868
26BF            869
26C1            870
26C3            871
26C5            872
26C7            873
26C9            874
26CB            875
26CD            876
26CF            877
26D1            878
26D3            879
26D5            880
26D7            881
26D9            882
26DB            883
26DD            884
26DF            885
26E1            886
26E3            887
26E5            888
26E7            889
26E9            890
26EB            891
26ED            892
26EF            893
26F1            894
26F3            895
26F5            896
26F7            897
26F9            898
26FB            899
26FD            900
26FF            901
2701            902
2703            903
2705            904
2707            905
2709            906
270B            907
270D            908
270F            909
2711            910
2713            911
2715            912
2717            913
2719            914
271B            915
271D            916
271F            917
2721            918
2723            919
2725            920
2727            921
2729            922
272B            923
272D            924
272F            925
2731            926
2733            927
2735            928
2737            929
2739            930
273B            931
273D            932
273F            933
2741            934
2743            935
2745            936
2747            937
2749            938
274B            939
274D            940
274F            941
2751            942
2753            943
2755            944
2757            945
2759            946
275B            947
275D            948
275F            949
2761            950
2763            951
2765            952
2767            953
2769            954
276B            955
276D            956
276F            957
2771            958
2773            959
2775            960
2777            961
2779            962
277B            963
277D            964
277F            965
2781            966
2783            967
2785            968
2787            969
2789            970
278B            971
278D            972
278F            973
2791            974
2793            975
2795            976
2797            977
2799            978
279B            979
279D            980
279F            981
27A1            982
27A3            983
27A5            984
27A7            985
27A9            986
27AB            987
27AD            988
27AF            989
27B1            990
27B3            991
27B5            992
27B7            993
27B9            994
27BB            995
27BD            996
27BF            997
27C1            998
27C3            999
27C5            1000

```

ASSEMBLY COMPLETED, NO ERROR(S) FOUND.

LINK LOC OBJECT TIME SOURCE STATEMENT

CONTROL'S SPECIFIED IN INVOCATION COMMAND MODS

OBJECT LIFE: E3 H20DHA DB7

SOURCE LIFE: E3 H20DHA VAP

SECTER-III MCR-29 MV-50 V22EWB7EN AT 0

V22 EWB7EN MV-50

## A.6. PWM using the HSO

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F3:HSODRV.A96

OBJECT FILE: :F3:HSODRV.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

| ERR | LOC           | OBJECT | LINE | SOURCE STATEMENT   |
|-----|---------------|--------|------|--|
|     |               |        | 1    | \$TITLE('HSODRV.A96: Driver module for HSO PWM program') |
|     |               |        | 2    |  |
|     |               |        | 3    | HSODRV MODULE MAIN, STACKSIZE(8)                         |
|     |               |        | 4    |  |
|     |               |        | 5    |  |
|     |               |        | 6    | PUBLIC HSO_ON_0, HSO_OFF_0                               |
|     |               |        | 7    | PUBLIC HSO_ON_1, HSO_OFF_1                               |
|     |               |        | 8    | PUBLIC HSO_TIME, HSO_COMMAND                             |
|     |               |        | 9    | PUBLIC SP, TIMER1, IOSO                                  |
|     |               |        | 10   |  |
|     |               |        | 11   | \$INCLUDE(DEMO96.INC)                                    |
|     |               | =1     | 12   | \$nolist ; Turn listing off for include file             |
|     |               | =1     | 60   | ; End of include file                                    |
|     |               |        | 61   |  |
|     | 0028          |        | 62   | rseg at 28H  |
|     |               |        | 63   |  |
|     |               |        | 64   | EXTRN OLD_STAT :byte                                     |
|     |               |        | 65   |  |
|     | 0028          |        | 66   | HSO_ON_0: dsw 1  |
|     | 002A          |        | 67   | HSO_OFF_0: dsw 1   |
|     | 002C          |        | 68   | HSO_ON_1: dsw 1  |
|     | 002E          |        | 69   | HSO_OFF_1: dsw 1   |
|     | 0030          |        | 70   | count: dsb 1   |
|     | 0037          |        | 71   | END  |
|     | 2080          |        | 72   | cseg at 2080H  |
|     | 2085          |        | 73   | DB 00H   |
|     | 208D          |        | 74   | EXTRN wait :entry  |
|     | 208D          |        | 75   | INCLUDES   |
|     | 2080 FA       |        | 76   | strt: DI   |
|     | 2081 A1000118 |        | 77   | LD SP, #100H   |
|     | 2085 510F1500 | E      | 78   | ANDB OLD_STAT, IOSO, #0FH                                |
|     | 2089 950F00   | E      | 79   | XORB OLD_STAT, #0FH                                      |
|     |               |        | 80   |  |
|     | 208C 5008     |        | 81   | initial: BX  |
|     | 208C A1000122 |        | 82   | LD CX, #0100H  |
|     | 2090 40545050 |        | 83   | END  |
|     | 2090 A100101C |        | 84   | loop: LD AX, #1000H                                      |
|     | 2094 48221C20 |        | 85   | SUB BX, AX, CX   |
|     | 2098 A0221C   |        | 86   | LD AX, CX  |
|     | 209D 345006   |        | 87   | MOVB 00H, 00H  |
|     | 209D 385E03   |        | 88   | MOVB 00H, 00H  |



[illegible]

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F3:HSOMOD.A96

OBJECT FILE: :F3:HSOMOD.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

```
ERR LOC  OBJECT      LINE      SOURCE STATEMENT
1      $TITLE('HSOMOD.A96: 8096 PWM PROGRAM MODIFIED FOR DRIVER')
2      $PAGEWIDTH(130)
3
4      ; This program will provide 3 PWM outputs on HSO pins 0-2
5      ; The input parameters passed to the program are:
6      ;
7      ;           HSO_ON_N      HSO on time for pin N
8      ;           HSO_OFF_N     HSO off time for pin N
9
10     ;           Where: Times are in timer1 cycles
11     ;           N takes values from 0 to 3
12
13     ;
14     ;
15
16     ; NOTE: Use this file to replace the declaration section of
17     ; the HSO PWM program from "$INCLUDE(DEMO96.INC)" through
18     ; the line prior to the label "wait". Also change the last
19     ; branch in the program to a "RET".
20
21     0000      21      RSEG
22
23     0000      23      D_STAT:          DSB          1
24     extrn     24      HSO_ON_0 :word , HSO_OFF_0 :word
25     extrn     25      HSO_ON_1 :word , HSO_OFF_1 :word
26     extrn     26      HSO_TIME :word , HSO_COMMAND :byte
27     extrn     27      TIMER1  :word , IOSO          :byte
28     extrn     28      SP      :word
29
30     public    29      OLD_STAT
31     OLD_STAT: 31      dsb          1
32     NEW_STAT: 32      dsb          1
33
34     INC       34      CX
35
36     0000      35      cseg
37     PUBLIC    36      wait
38
39     0000      37      wait: JBS     IOSO, 6, wait      ; Loop until HSO holding register
40     NOP       38      ; is empty
41
42     0000      39      ; For operation with interrupts 'store_stat:' would be the
43     ; entry point of the routine.
44     ; Note that a DI or PUSHF might have to be added.
```

```

0004                                45 store_stat:
0004 510F0002                        E 46         ANDB NEW_STAT, IOS0, #0FH      ; Store new status of HSO
0008 980201                        R 47         CMPB OLD_STAT, NEW_STAT
000B DFF3                          48         JE wait
000D 940201                        R 49         XORB OLD_STAT, NEW_STAT
                                50
                                51
0010                                52 check_0:
0010 300113                        R 53         JBC OLD_STAT, 0, check_1      ; Jump if OLD_STAT(0)=NEW_STAT(0)
0013 380209                        R 54         JBS NEW_STAT, 0, set_off_0
                                55
0016                                56 set_on_0:
0016 B13000                        E 57         LDB HSO_COMMAND, #00110000B      ; Set HSO for timer1, set pin 0
0019 44000000                      E 58         ADD HSO_TIME, TIMER1, HSO_OFF_0      ; Time to "set" pin = Timer1 value
001D 2007                          59         BR check_1      ; + Time for pin to be low
                                60
001F                                61 set_off_0:
001F B11000                        E 62         LDB HSO_COMMAND, #00010000B      ; Set HSO for timer1, clear pin 0
0022 44000000                      E 63         ADD HSO_TIME, TIMER1, HSO_ON_0      ; Time to clear pin = Timer1 value
                                64         ; + Time for pin to be high
0026                                65 check_1:
0026 310113                        R 66         JBC OLD_STAT, 1, check_done      ; Jump if OLD_STAT(1)=NEW_STAT(1)
0029 390209                        R 67         JBS NEW_STAT, 1, set_off_1
                                68
002C                                69 set_on_1:
002C B13100                        E 70         LDB HSO_COMMAND, #00110001B      ; Set HSO for timer1, set pin 1
002F 44000000                      E 71         ADD HSO_TIME, TIMER1, HSO_OFF_1      ; Time to set pin = Timer1 value
0033 2007                          72         BR check_done
                                73
0035                                74 set_off_1:
0035 B11100                        E 75         LDB HSO_COMMAND, #00010001B      ; Set HSO for timer1, clear pin 1
0038 44000000                      E 76         ADD HSO_TIME, TIMER1, HSO_ON_1      ; Time to clear pin = Timer1 value
                                77         ; + Time for pin to be high
003C                                78 check_done:
003C B00201                        R 79         LDB OLD_STAT, NEW_STAT      ; Store current status and
                                80         ; wait for interrupt flag
                                81
                                82 RET
                                83 use "BR wait" if this routine is used with the driver
                                84
0040                                85 END

```

ASSEMBLY COMPLETED, NO ERROR(S) FOUND.

SERIES-111 MCS-96 MACRO ASSEMBLER AT 0

AT 20000000

## A.7. Serial Port

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: F3:SP.A96

OBJECT FILE: F3:SP.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

| ERR LOC | OBJECT | LINE | SOURCE STATEMENT                                       |
|---------|--------|------|--|
|         |        | 1    |  |
|         |        | 2    | \$TITLE('SP.A96: SERIAL PORT DEMO PROGRAM')            |
|         |        | 3    |  |
|         |        | 4    | \$INCLUDE(DEMO96.INC)                                  |
|         |        | 5    | \$nolist ; Turn listing off for include file           |
|         |        | 53   | ; End of include file                                  |
|         |        | 54   |  |
|         |        | 55   | rseg at 28H  |
|         |        | 56   |  |
|         |        | 57   | CHR: dsb 1   |
|         |        | 58   | SPTIME: dsb 1  |
|         |        | 59   | TEMPO: dsb 1   |
|         |        | 60   | TEMP1: dsb 1   |
|         |        | 61   | RCV_FLAG: dsb  |
|         |        | 62   |  |
|         |        | 63   | cseg at 200CH  |
|         |        | 64   |  |
|         |        | 65   | DCW ser_port_int                                       |
|         |        | 66   |  |
|         |        | 67   | cseg at 2080H  |
|         |        | 68   |  |
|         |        | 69   | LD SP, #100H   |
|         |        | 70   |  |
|         |        | 71   | LDB IOC1, #00100000B ; Set P2.0 to TXD                 |
|         |        | 72   |  |
|         |        | 73   | YDD H20 ; Baud rate = input frequency / (64*baud_val)  |
|         |        | 74   | YDD H20 ; baud_val = (input frequency/64) / baud rate  |
|         |        | 75   |  |
|         |        | 76   |  |
|         |        | 77   | baud_val equ 39 ; 39 = (12,000,000/64)/4800 baud       |
|         |        | 78   |  |
|         |        | 79   | BAUD_HIGH equ ((baud_val-1)/256) OR 80H ; Set MSB to 1 |
|         |        | 80   | BAUD_LOW equ (baud_val-1) MOD 256                      |
|         |        | 81   |  |
|         |        | 82   |  |
|         |        | 83   | LDB BAUD_REG, #BAUD_LOW                                |
|         |        | 84   | LDB BAUD_REG, #BAUD_HIGH                               |
|         |        | 85   |  |



```

20BD B14911      86          LDB      SPCON, #01001001B      ; Enable receiver, Mode 1
87
88
89
90
91          STB      SBUF, CHR      ; Clear serial Port
92          LDB      TEMPO, #00100000B      ; Set TI-temp
93
94          LDB      INT_MASK, #01000000B      ; Enable Serial Port Interrupt
95
96          EI
97          loop:    BR      loop      ; Wait for serial port interrupt
98
99          ser_port_int:
100         PUSHF
101         rd_again:
102         LDB      SPTEMP, SPSTAT      ; This section of code can be replaced
103         ORB      TEMPO, SPTEMP      ; with "ORB TEMPO, SP_STAT" when the
104         ANDB      SPTEMP, #01100000B      ; serial port TI and RI bugs are fixed
105         JNE      rd_again      ; Repeat until TI and RI are properly cleared
106
107         get_byte:
108         JBC      TEMPO, 6, put_byte      ; If RI-temp is not set
109         STB      SBUF, CHR      ; Store byte
110         ANDB      TEMPO, #10111111B      ; CLR RI-temp
111         LDB      RCV_FLAG, #0FFH      ; Set bit-received flag
112
113         put_byte:
114         JBC      RCV_FLAG, 0, continue      ; If receive flag is cleared
115         JBC      TEMPO, 5, continue      ; If TI was not set
116         LDB      SBUF, CHR      ; Send byte
117         ANDB      TEMPO, #11011111B      ; CLR TI-temp
118
119         ANDB      CHR, #01111111B      ; This section of code appends
120         CMPB     CHR, #0DH      ; an LF after a CR is sent
121         JNE      clr_rcv
122         LDB      CHR, #0AH
123         BR      continue
124
125         clr_rcv:
126         CLR      RCV_FLAG      ; Clear bit-received flag
127
128         continue:
129         POPF
130         RET
131
132         END
20D1

```

ASSEMBLY COMPLETED, NO ERROR(S) FOUND.

# A.8. A to D Converter

SOURCE FILE: :F3:ATOD.A96

OBJECT FILE: :F3:ATOD.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

| ERR LOC | OBJECT        | LINE | SOURCE STATEMENT  |
|---------|---------------|------|---|
|         |               | 1    | \$TITLE('ATOD.A96: SCANNING THE A TO D CHANNELS')               |
|         |               | 2    |   |
|         |               | 3    | \$INCLUDE(DEMO96.INC)   |
|         |               | 4    | \$nolist ; Turn listing off for include file                    |
|         |               | 52   | End of include file   |
|         |               | 53   |   |
|         | 002B          | 54   | RSEG at 28H   |
|         |               | 55   |   |
|         | 0020          | 56   | BL EQU BX: BYTE   |
|         | 001E          | 57   | DL EQU DX: BYTE   |
|         |               | 58   |   |
|         | 002B          | 59   | RESULT_TABLE:   |
|         | 002B          | 60   | RESULT_1: dsw 1   |
|         | 002A          | 61   | RESULT_2: dsw 1   |
|         | 002C          | 62   | RESULT_3: dsw 1   |
|         | 002E          | 63   | RESULT_4: dsw 1   |
|         |               | 64   |   |
|         | 2080          | 65   |   |
|         |               | 66   | cseg at 2080H   |
|         |               | 67   |   |
|         | 2080 A1000118 | 68   |   |
|         | 2084 0120     | 69   | start: LD SP, #100H ; Set Stack Pointer                         |
|         |               | 70   | CLR BX  |
|         | 2086 55082002 | 71   |   |
|         |               | 72   | next: ADDB AD_COMMAND, BL, #1000B ; Start conversion on channel |
|         |               | 73   | ; indicated by BL register                                      |
|         |               | 74   |   |
|         | 208A FD       | 75   | NOP ; Wait for conversion to start                              |
|         | 208B FD       | 76   | NOP   |
|         | 208C 3B02FD   | 77   | check: JBS AD_RESULT_LO, 3, check ; Wait while A to D is busy   |
|         |               | 78   |   |
|         | 208F B0021C   | 79   | LDB AL, AD_RESULT_LO ; Load low order result                    |
|         | 2092 B0031D   | 80   | LDB AH, AD_RESULT_HI ; Load high order result                   |
|         |               | 81   |   |
|         | 2095 5420201E | 82   | ADDB DL, BL, BL ; DL=BL*2                                       |
|         | 2099 AC1E1E   | 83   | LDBZE DX, DL  |
|         | 209C C31E281C | 84   | ST AX, RESULT_TABLE[DX] ; Store result indexed by BL*2          |
|         |               | 85   |   |
|         | 20A0 1720     | 86   | INCB BL ; Increment BL modulo 4                                 |



## APPENDIX B HSO AND A TO D UNDER INTERRUPT CONTROL

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F3:A2DHSO.A96

OBJECT FILE: :F3:A2DHSO.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

| ERR | LOC  | OBJECT | LINE | SOURCE STATEMENT   |
|-----|------|--------|------|--|
|     |      |        | 1    | \$TITLE ('A2DHSO.A96: GENERATING PWM OUTPUTS FROM A TO D INPUTS')  |
|     |      |        | 2    |  |
|     |      |        | 3    | ; This program will provide 3 PWM outputs on HSO pins 0-2          |
|     |      |        | 4    | ; and one on the PWM.  |
|     |      |        | 5    |  |
|     |      |        | 6    | ; The PWM values are determined by the input to the A/D converter. |
|     |      |        | 7    |  |
|     |      |        | 8    | ;;;                  |
|     |      |        | 9    |  |
|     |      |        | 10   | \$INCLUDE(DEMO96.INC)  |
| =1  |      |        | 11   | \$nolist ; Turn listing off for include file                       |
| =1  |      |        | 59   | ; End of include file  |
|     |      |        | 60   |  |
|     | 002B |        | 61   | RSEG AT 28H  |
|     |      |        | 62   |  |
|     | 001E |        | 63   | DL EQU DX:BYTE   |
|     |      |        | 64   |  |
|     | 002B |        | 65   | ON_TIME:   |
|     | 002B |        | 66   | PWM_TIME_1: DSW 1  |
|     | 002A |        | 67   | HSO_ON_0: DSW 1  |
|     | 002C |        | 68   | HSO_ON_1: DSW 1  |
|     | 002E |        | 69   | HSO_ON_2: DSW 1  |
|     |      |        | 70   |  |
|     | 0030 |        | 71   | RESULT_TABLE:  |
|     | 0030 |        | 72   | RESULT_0: DSW 1  |
|     | 0032 |        | 73   | RESULT_1: DSW 1  |
|     | 0034 |        | 74   | RESULT_2: DSW 1  |
|     | 0036 |        | 75   | RESULT_3: DSW 1  |
|     |      |        | 76   |  |
|     | 003B |        | 77   | NXT_ON_T: DSW 1  |
|     | 003A |        | 78   | NXT_OFF_0: DSW 1   |
|     | 003C |        | 79   | NXT_OFF_1: DSW 1   |
|     | 003E |        | 80   | NXT_OFF_2: DSW 1   |
|     | 0040 |        | 81   | COUNT: DSL 1   |
|     | 0044 |        | 82   | AD_NUM: DSW 1  |
|     | 0046 |        | 83   | TMP: DSW 1   |
|     | 004B |        | 84   | HSO_PER: DSW 1   |
|     | 004A |        | 85   | LAST_LOAD: DSW 1   |
|     |      |        | 86   |  |

Channel being converted

intel®

MCS-96

68000



```

2000 116D0E      181 87  cseg  HDB AT 2000H      ; C1000 61 1
2000 8020      180 88  ;
2002 1D2194V    188 89          DCW      start      ; Timer_ovf_int
2004 8020      188 90  HDB  DCW  [V01] Atod_done_int ;
2006 CC20E0+    188 91          DCW      start      ; HSI_data_int
2008 8020      188 92  FD   DCW  H20 HSD_exec_int
200A 8020      188 93  FDB  H20 COM_VMB #00001000B ; 202 H20 604 210621 C1000 610 5
200C 8020      188 94  $EJECT H21 001 5 H21 001 1 H20 001 5
200E 8020      188 95  H20
2010 8020      188 96  cseg  AT 2080H
2012 8020      188 97  FD   H20 TIME H21 001 1
2014 A100011B    188 98  start: LD  H20 SP, #100H ; Set Stack Pointer
2016 011C383C    188 99  CLR  H20 AX ;
2018 051C        188 100 wait: DEC AX ; wait approx. 0.2 seconds for
201A D7FC        188 101 JNE  wait ; SBE to finish communications
201C 8020      188 102 FD   H20 TIME H21 001 0
201E 114400F    188 103 FDB  CLR  H20 AD_NUM
2020 8020      188 104 VDB  H20 AX H21 001 0 H20 001 0
2022 A180002B    188 105 LD  H20 PWM_TIME_1, #080H
2024 A100014B    188 106 LD  H20 HSD_PER, #100H
2026 A140002A    188 107 LD  H20 HSD_ON_0, #040H
2028 A180002C    188 108 LD  H20 HSD_ON_1, #080H
202A A1C0002E    188 109 LD  H20 HSD_ON_2, #0C0H
202C 8020      188 110 BM   ;
202E 4500010A3B  188 111 ADD  H20 NXT_ON_T, Timer1, #100H ;
2030 8020      188 112 FDB  H20 COM_VMB #00001000B ;
2032 B13606      188 113 LDB  H20 HSD_COMMAND, #00110110B ; Set HSD for timer1, set pin 0,1
2034 A03804V    188 114 LDB  H20 HSD_TIME, NXT_ON_T ; with interrupts
2036 FD        188 115 NOP
2038 FD0380+    188 116 FD   H20 TIME H21 001 1
203A B12206+    188 117 FDB  LDB  H20 HSD_COMMAND, #00100010B ; Set HSD for timer1, set pin 2
203C 643804      188 118 ADD  H20 HSD_TIME, NXT_ON_T ; without interrupt
203E 8020      188 119 H06
2040 91074A      188 120 ORB  H20 LAST_LOAD, #00000111B ; Last loaded value was set all pins
2042 B10A08V    188 121 LDB  H20 INT_MASK, #00001010B ; Enable HSD and A/D interrupts
2044 B10A09B    188 122 VDB  LDB  H20 INT_PENDING, #00001010B ; Fake an A/D and HSD interrupt
2046 FB        188 123 EI
2048 8020      188 124
204A 91010F      188 125 loop: ORB H20 Port1, #00000001B ; set P1.0
204C 65010040    188 126 ADD  H20 COUNT, #01
204E A40042V    188 127 ADDC H20 COUNT+2, zero
2050 71FE0F      188 128 ANDB H20 Port1, #11111110B ; clear P1.0
2052 27F130E    188 129 BR  H20 loop
2054 8020      188 130
2056 8020      188 131 $EJECT
2058 8020      188 132
205A 8020      188 133
205C 8020      188 134
205E 8020      188 135

```

```

132
133
134
135
136
137 HSO_exec_int:
138 130 PUSHF
139 138 ORB 88 Port1, #00000010B ; Set p1.1
140 158
141 151 SUB VDDC TMP, TIMER1, NXT_ON_T
142 159 CMP VDDC TMP, ZERO
143 152 JLT JNB set_off_times, #0000001B
144 154
145 155 set_on_times:
146 155 ADD NXT_ON_T, HSO_PER
147 151 LDB HSO_COMMAND, #00110110B ; Set HSO for timer1, set pin 0,1
148 150 LD HSO_TIME, NXT_ON_T
149 118 NOP
150 118 NOP
151 111 LDB HSO_COMMAND, #00100010B ; Set HSO for timer1, set pin 2
152 119 LD HSO_TIME, NXT_ON_T
153 112
154 114 ORB LD LAST_LOAD, #00000111B ; Last loaded value was all ones
155 113
156 115 LDB PWM_CONTROL, PWM_TIME_1 ; Now is as good a time as any
157 111 to update the PWM reg
158 110 BR check_done
159 104
160 108
161 108 set_off_times:
162 109 JBC LAST_LOAD, 0, check_done
163 109
164 104 ADD NXT_OFF_0, NXT_ON_T, HSO_ON_0
165 103 LDB HSO_COMMAND, #00010000B ; Set HSO for timer1, clear pin 0
166 105 LD HSO_TIME, NXT_OFF_0
167 101
168 100 NOP
169 88 ADD NXT_OFF_1, NXT_ON_T, HSO_ON_1
170 88 LDB HSO_COMMAND, #00010001B ; Set HSO for timer1, clear pin 1
171 81 LD HSO_TIME, NXT_OFF_1
172 89
173 82 NOP
174 84 ADD NXT_OFF_2, NXT_ON_T, HSO_ON_2
175 83 LDB HSO_COMMAND, #00010010B ; Set HSO for timer1, clear pin 2
176 85 LD HSO_TIME, NXT_OFF_2
177 81
178 80 ANDB LAST_LOAD, #11111000B ; Last loaded value was all 0s
179 84
180 81 check_done:
181 81 ANDB Port1, #11111101B ; Clear P1.1

```



**MCS®-96**



## APPENDIX C SOFTWARE SERIAL PORT

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F3:SWPORT.A96

OBJECT FILE: :F3:SWPORT.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

\*\*\*\*\* A COMPILED \*\*\*\*\* NO ERROR(S) FOUND

| ERR LOC | OBJECT | LINE   | SOURCE STATEMENT   |
|---------|--------|--------|--|
|         |        | 551 1  | \$TITLE('SWPORT.A96 : SOFTWARE IMPLEMENTED ASYNCHRONOUS SERIAL PORT')  |
|         |        | 550 2  |  |
|         |        | 551 3  | ; This module provides a software implemented asynchronous serial port |
|         |        | 551 4  | ; for the 8096. HSO.5 is used for transmit data. HSI.2 is used for     |
|         |        | 551 5  | ; receive data. Note: the choice of HSO.5 and HSI.2 is arbitrary).     |
|         |        | 551 6  |  |
|         |        | 551 7  | \$INCLUDE(DEMO96.INC)  |
|         |        | 551 8  | \$nolist ; Turn listing off for include file                           |
|         |        | 551 56 | ; End of include file  |
|         |        | 551 57 |  |
|         |        | 551 58 | ; VARIABLES NEEDED BY THE SOFTWARE SERIAL PORT                         |
|         |        | 551 59 | =====  |
|         |        | 551 60 | ; \$1 ON LINE LDXI   |
|         |        | 551 61 | CFR8 rseg  |
|         |        | 551 62 |  |
|         |        | 551 63 | ios1_save: dsb 1 ; Used to save contents of ios1                       |
|         |        | 551 64 | rcve_state: dsb 1  |
|         |        | 551 65 | rxrdy equ 1 ; indicates receive done                                   |
|         |        | 551 66 | rxovrrun equ 2 ; indicates receive overflow                            |
|         |        | 551 67 | rip equ 4 ; receive in progress flag                                   |
|         |        | 551 68 | rcve_buf: dsb 1000 ; used to double buffer receive data                |
|         |        | 551 69 | rcve_reg: dsb 1 ; used to deserialize receive                          |
|         |        | 551 70 | sample_time: dsb 1 ; records last receive sample time                  |
|         |        | 551 71 |  |
|         |        | 551 72 | serial_out: dsb 1 ; Holds the output character + framing (start and    |
|         |        | 551 73 | stop bits) for transmit process.                                       |
|         |        | 551 74 | baud_count: dsb 1 ; Holds the period of one bit in units               |
|         |        | 551 75 | of T1 ticks.   |
|         |        | 551 76 | txd_time: dsb 1 ; Transition time of last Txd bit that was             |
|         |        | 551 77 | sent to the CAM  |
|         |        | 551 78 | char: dsb 1 ; for test only  |
|         |        | 551 79 |  |
|         |        | 551 80 | =====  |
|         |        | 551 81 | COMMANDS ISSUED TO THE HSD UNIT  |
|         |        | 551 82 | =====  |
|         | 0035   | 551 83 | mark_command equ 0110101b ; timer1.set.interrupt on 5                  |
|         | 0015   | 551 84 | space_command equ 0010101b ; timer1.clr.interrupt on 5                 |
|         | 0018   | 551 85 | sample_command equ 0011000b ; software timer 0                         |
|         |        | 551 86 |  |
|         |        | 551 87 | \$eject  |



|               |   |     |  |
|---------------|---|-----|--|
| 2080          |   | 88  | cseg at 2080h  |
|               |   | 89  |  |
| 2080          |   | 90  | reset_loc:   |
|               |   | 91  | ; The 8096 starts executing here on reset, the program will initialize the |
|               |   | 92  | ; the software serial port and run a simple test to exercise it.           |
|               |   | 93  |  |
| 2080 FA       |   | 94  | di   |
| 2081 A1F0001B |   | 95  | ld sp,#0F0h  |
| 2085 C9C012   |   | 96  | push #4800   |
| 208B EF0000   | R | 97  | call setup_serial_port   |
| 208B B16C0B   |   | 98  | ldb int_mask,#01101100b ; serial, swt, hso, hsi                            |
| 20BE FB       |   | 99  | ei   |
|               |   | 100 |  |
|               |   | 101 |  |
| 20BF          |   | 102 | test1:   |
|               |   | 103 | ; A simple test of the serial port routines.                               |
|               |   | 104 | ; While no characters are received an incrementing pattern is sent to the  |
|               |   | 105 | ; serial output. When a character is received the incrementing pattern     |
|               |   | 106 | ; "jumps" to the character received and proceeds from there.               |
|               |   | 107 |  |
|               |   | 108 | CR equ ODH ; Carriage return   |
| 20BF B10DOC   | R | 109 | ldb char,#CR   |
| 2092          |   | 110 | testiloop:   |
| 2092 AC0C1C   | R | 111 | ldbz ax,char   |
| 2095 C81C     |   | 112 | push ax  |
| 2097 EF3000   | R | 113 | call char_out  |
|               |   | 114 |  |
| 209A 990DOC   | R | 115 | cmpb char,#CR ; Pause on Carriage return                                   |
| 209D D706     |   | 116 | bne nopause  |
| 209F 011C     |   | 117 | clr ax   |
| 20A1          |   | 118 | pause:   |
| 20A1 071C     |   | 119 | inc ax   |
| 20A3 D7FC     |   | 120 | bne pause  |
| 20A5          |   | 121 | nopause:   |
|               |   | 122 |  |
| 20A5 170C     | R | 123 | incb char  |
| 20A7          |   | 124 | test2:   |
| 20A7 EF4400   | R | 125 | call csts ; char ready?  |
| 20AA 98001C   |   | 126 | cmpb al,0  |
| 20AD DFE3     |   | 127 | be testiloop ; loop if not   |
| 20AF EF4C00   | R | 128 | call char_in   |
| 20B2 B01C0C   | R | 129 | ldb char,al  |
| 20B5 27DB     |   | 130 | br testiloop   |
|               |   | 131 | \$eject  |
|               |   | 132 |  |
|               |   | 133 |  |
|               |   | 134 |  |
|               |   | 135 |  |

```

0000                                132
                                133      cseg
                                134
0000                                135      setup_serial_port:
                                136      ; Called on system reset to initiate the software serial port.
                                137
0000 CC22                            138      pop      cx      ; the return address
0002 CC20                            139      pop      bx      ; the baud rate (in decimal)
0004 A107001E                        140      ld      dx, #0007h ; dx:ax:=500,000 (assumes 12 Mhz crystal)
0008 A120A11C                        141      ld      ax, #0A120h
000C 8C201C                          142      divu    ax, bx      ; calculate the baud count (500,000/baudrate)
000F C0081C                          R 143      st      ax, baud_count
0012 C00600                          R 144      st      0, serial_out ; clear serial out
0015 B16016                          145      ldb     ioc1, #01100000b ; Enable HSD.5 and Txd
0018 3E15FD                          146      bbs     ios0, 6, $ ; Wait for room in the HSD CAM
                                147      ; and issue a MARK command.
001B 44140A0A                        R 148      add     txd_time, timer1, 20
001F B13506                          149      ldb     hso_command, #mark_command
0022 A00A04                          R 150      ld      hso_time, txd_time
0025 1102                            R 151      clrb    rcve_buf ; clear out the receive variables
0027 1103                            R 152      clrb    rcve_reg
0029 1101                            R 153      clrb    rcve_state
002B EF4800                          154      call   init_receive ; setup to detect a start bit
002E E322                            155      br      [cx] ; return
                                156      $eject
                                157
0030                                158      char_out:
                                159      ; Output character to the software serial port
                                160
0030 CC22                            161      pop      cx      ; the return address
0032 CC20                            162      pop      bx      ; the character for output
0034 B10121                          163      ldb     (bx+1), #01h ; add the start and stop bits
0037 642020                          164      add     bx, bx ; to the char and leave as 16 bit
003A                                165      wait_for_xmit:
003A 880006                          R 166      cmp     serial_out, 0 ; wait for serial_out=0 (it will be cleared by
003D D7FB                            167      bne     wait_for_xmit ; the hso interrupt process)
003F C00620                          R 168      st      bx, serial_out ; put the formatted character in serial_out
0042 E322                            169      br      [cx] ; return to caller
                                170
0044                                171      csts:
                                172      ; Returns "true" (ax<>0) if char_in has a character.
                                173
0044 011C                            174      clr     ax
0046 300102                          R 175      bbs     rcve_state, 0, csts_exit
0049 071C                            176      inc     ax
004B                                177      csts_exit:
004B F0                              178      ret
                                179
004C                                180      char_in:

```

```

00C1 BACC      181      ; Get a character from the software serial port
00B0 27C0100   182      ;
00B0 11C00     183      ; wait for character ready
004C 3001FD    R      184      bbc      rcve_state,0,char_in
004F F2        185      pushf      ; set up a critical region
0050 71FE01    R      186      andb      rcve_state,#not(rxrdy)
0053 AC021C    R      187      ldbze     al,rcve_buf
0056 F3        188      popf      ; leave the critical region
0057 F0        189      ret
190      $eject
191      ;
0058          192      hso_isr:
193      ; Fields the hso interrupts and performs the serialization of the data.
194      ; Note: this routine would be incorporated into the hso service strategy
195      ; for an actual system.
196      ;
2006 20        197      cseg      at 2006h
2006 5800      R      198      dcw      hso_isr      ; Set up vector
199      ;
0058          200      cseg
0058 F2        201      pushf
0059 64080A    R      202      add      txd_time,baud_count
005C 8B0006    R      203      cmp      serial_out,0 ; if character is done send a mark
005F DF0D      204      be      send_mark
0061 0B0106    R      205      shr      serial_out,#1 ; else send bit 0 of serial_out and shift
0064 DB0B      206      bc      send_mark ; serial_out left one place
0066          207      send_space:
0066 B11506      208      ldb      hso_command,#space_command
0069 A00A04    R      209      ld      hso_time,txd_time
006C 2006      210      br      hso_isr_exit
006E          211      send_mark:
006E B13506      212      ldb      hso_command,#mark_command
0071 A00A04    R      213      ld      hso_time,txd_time
214      ;
0074          215      hso_isr_exit:
0074 F3        216      popf
0075 F0        217      ret
218      $eject
219      ;
0076          220      init_receive:
221      ; Called to prepare the serial input process to find the leading edge of
222      ; a start bit.
223      ;
0076 B10015      224      ldb      ioc0,#00000000b ; disconnect change detector
0079 B12003      225      ldb      hsi_mode,#00100000b ; negative edges on HSI.2
007C          226      flush_fifo:
007C 901600    R      227      orb      ios1_save,ios1
007F 37000B    R      228      bbc      ios1_save,7,flush_fifo_done
0082 B0061C      229      ldb      al,hsi_status
0085 A0041C      230      ld      ax,hsi_time ; trash the fifo entry

```

```

008B 717F00      R    231      andb    ios1_save,#not(80h)      ; clear bit 7.
008B 27EF        232      br        flush_fifo
008D            233      flush_fifo_done:
008D B11015      R    234      ldb     ioc0,#00010000b      ; connect HSI.2 to detector
0090 F0          235      ret
0091            236
0091            237
0091            238
0091            239      hsi_isr:
0091            240      ; Fields interrupts from the HSI unit, used to detect the leading edge
0091            241      ; of the START bit
0091            242      ; Note: this routine would be incorporated into the HSI strategy of an actual
0091            243      ; system.
0091            244      ;
0091            245      cseg at 2004h
0091 2004 9100      R    246      dcw     hsi_isr      ; setup the interrupt vector
0091            247
0091            248      cseg
0091 0091 F2        R    249      pushf
0091 0092 C81C      R    250      push    ax
0091 0094 B0061C    R    251      ldb     al,hsi_status
0091 0097 A00404    R    252      ld      sample_time,hsi_time
0091 009A 341C15    R    253      bbc     al,4,exit_hsi
0091 009D 3F15FD    R    254      bbs     ios0,7,$      ; wait for room in HSO holding reg
0091 00A0 A0081C    R    255      ld      ax,baud_count      ; send out sample command in 1/2
0091 00A3 08011C    R    256      shr     ax,#1      ; bit time
0091 00A6 641C04    R    257      add     sample_time,ax
0091 00A9 B11806    R    258      ldb     hso_command,#sample_command
0091 00AC C00404    R    259      st      sample_time,hso_time
0091 00AF B10015    R    260      ldb     ioc0,#00000000b      ; disconnect hsi.2 from change detector
0091 00B2          261      exit_hsi:
0091 00B2 CC1C      R    262      pop     ax
0091 00B4 F3        R    263      popf
0091 00B5 F0        R    264      ret
0091            265      $eject
0091            266
0091            267      software_timer_isr:
0091            268      ; Fields the software timer interrupt, used to deserialize the incoming data.
0091            269      ; Note: this routine would be incorporated into the software timer strategy
0091            270      ; in an actual system.
0091            271      ;
0091            272      cseg at 200Ah
0091 200A B600      R    273      dcw     software_timer_isr      ; setup vector
0091            274
0091 00B6          275      cseg
0091 00B6 F2        R    276      pushf
0091 00B7 901600    R    277      orb     ios1_save,ios1
0091 00BA 71FE00    R    278      andb    ios1_save,#not(01h)      ; clear bit 0
0091 00BD 51FC0100  R    279      andb    0,rcv_state,#0fch      ; All bits except rxrdy and overrun=0
0091 00C1 D70C      R    280      bne     process_data

```



```

00C3      281  process_start_bit:
00C3 350604 282      bbs     hsi_status,5,start_ok
00C6 2FAE   283      call    init_receive
00C8 2032   284      br      software_timer_exit
00CA      285  start_ok:
00CA 910401 R 286      orb     rcve_state,#rip ; set receive in progress flag
00CD 2021   287      br      schedule_sample
00CF      288
00CF      289  process_data:
00CF 3F010E R 290      bbs     rcve_state,7,check_stopbit
00D2 180103 R 291      shrb    rcve_reg,#1
00D5 350603   292      bbs     hsi_status,5,datazero
00D8 918003 R 293      orb     rcve_reg,#80h ; set the new data bit
00DB      294  datazero:
00DB 751001 R 295      addb    rcve_state,#10h ; increment bit count
00DE 2010   296      br      schedule_sample
00E0      297
00E0      298  check_stopbit:
00E0 3506FD   299      bbs     hsi_status,5,$ ; DEBUG ONLY
00E3 B00302 R 300      ldb     rcve_buf,rcve_reg
00E6 910101 R 301      orb     rcve_state,#rxrdy
00E9 710301 R 302      andb    rcve_state,#03h ; Clear all but ready and overrun bits
00EC 2F8B   303      call    init_receive
00EE 200C   304      br      software_timer_exit
00F0      305
00F0      306  schedule_sample:
00F0 3F15FD   307      bbs     ios0,7,$ ; wait for holding reg empty
00F3 B11806   308      ldb     hso_command,#sample_command
00F6 640804 R 309      add     sample_time,baud_count
00F9 C00404 R 310      st      sample_time,hso_time
00FC      311
00FC      312  software_timer_exit:
00FC F3      313      popf
00FD F0      314      ret
00FE      315
00FE      316
00FE      317      end

```

ASSEMBLY COMPLETED. NO ERROR(S) FOUND.

CONTROLS SPECIFIED IN INVOCATION COMMAND: NONE

OBJECT LIFE: 43 MILLISEC DB7

SOURCE LIFE: 43 MILLISEC VAP

SERIES-III MCS-8P M4C80 VER20W8F01 A1 0

MOTOR CONTROL PROGRAM  
APPENDIX D

## APPENDIX D MOTOR CONTROL PROGRAM

SERIES-III MCS-96 MACRO ASSEMBLER, V1 0

SOURCE FILE: :F3:MOTCON.A96

OBJECT FILE: :F3:MOTCON.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

| ERR | LOC | OBJECT | LINE | SOURCE STATEMENT   |
|-----|-----|--------|------|--|
|     |     |        | 1    | \$TITLE ('MOTCON.A96: Motor Control Example Program')                    |
|     |     |        | 2    |  |
|     |     |        | 3    | USE WITH C-STEP or later parts   |
|     |     |        | 4    |  |
|     |     |        | 5    | December 20, 1984  |
|     |     |        | 6    |  |
|     |     |        | 7    | \$INCLUDE(DEMO96.INC)  |
|     |     |        | 8    | \$nolist ; Turn listing off for include file                             |
|     |     |        | 9    | ; End of include file  |
|     |     |        | 10   |  |
|     |     |        | 11   | Initial Values   |
|     |     |        | 12   |  |
|     |     |        | 13   | min_hsil_t equ 30 ; min period for PHA edges in mode1 before mode2       |
|     |     |        | 14   |  |
|     |     |        | 15   | min_hsi_t equ 2*min_hsil_t   |
|     |     |        | 16   | ; min period for PHA edges in mode0 before mode1                         |
|     |     |        | 17   |  |
|     |     |        | 18   | max_hsil_t equ 3*min_hsil_t + min_hsil_t/2                               |
|     |     |        | 19   | ; max period for PHA edges in mode1 before mode0                         |
|     |     |        | 20   |  |
|     |     |        | 21   | HSD0_dly_period equ 110 ; delay for HSD0 timer 0 (timed count of pulses) |
|     |     |        | 22   | ; min period for 5 T2 clocks before mode 1                               |
|     |     |        | 23   |  |
|     |     |        | 24   | swt1_dly_period equ 250 ; delay for software timer 1                     |
|     |     |        | 25   | swt2_dly_period equ 250 ; delay for software timer 2                     |
|     |     |        | 26   | max_power equ Offh   |
|     |     |        | 27   | max_brake equ Offh   |
|     |     |        | 28   | maximum_hold equ 080H  |
|     |     |        | 29   | brake_pnt equ 1200   |
|     |     |        | 30   | position_pnt equ 100   |
|     |     |        | 31   | velocity_pnt equ 16  |
|     |     |        | 32   |  |
|     |     |        | 33   | RSEG at 024H   |
|     |     |        | 34   |  |
|     |     |        | 35   | tmp: dsl 1   |
|     |     |        | 36   | timer_2: dsl 1   |

|               |     |                |                                    |
|---------------|-----|----------------|------------------------------------|
| 002C          | 86  | tmr2_old:      | dsl 1                              |
| 0030 816E10   | 87  | position:      | dsl 1                              |
| 0034 816E0E   | 88  | des_pos:       | dsl 1                              |
| 0038          | 89  | pos_err:       | dsl 1                              |
| 003C 05E9     | 90  | delta_p:       | dsl 1                              |
| 0040 88003C   | 91  | time:          | dsl 1                              |
| 0044 E098ED   | 92  | des_time:      | dsl 1                              |
| 0048 022C     | 93  | time_err:      | dsl 1                              |
| 5083 V130112C | 94  |                |                                    |
| 5081 119B     | 95  | \$EJECT        |                                    |
|               | 96  |                |                                    |
| 004C 816E1A   | 97  | last_time_err: | dsw 1                              |
| 004E V1E0001B | 98  | last_pos_err:  | dsw 1                              |
| 0050          | 99  | pos_delta:     | dsw 1                              |
| 0052          | 100 | time_delta:    | dsw 1                              |
| 0054          | 101 | last_pos:      | dsw 1                              |
| 0056          | 102 | last1_time:    | dsw 1                              |
| 0058          | 103 | last2_time:    | dsw 1                              |
| 005A          | 104 | boost:         | dsw 1                              |
| 005C          | 105 | tmp1:          | dsw 1                              |
| 005E          | 106 | out_ptr:       | dsw 1                              |
| 0060 1050     | 107 | offset:        | dsw 1                              |
| 0062 1050     | 108 | nxt_pos:       | dsw 1                              |
| 0064 5055     | 109 | rpwr:          | dsw 1                              |
| 0066 1050     | 110 | old_t2:        | dsw 1                              |
| 5009 8053     | 111 |                |                                    |
| 0068 0454     | 112 | direct:        | dsb 1 ; 1=forward, 0=reverse       |
| 0069 1050     | 113 | pwm_dir:       | dsb 1                              |
| 006A 0055     | 114 | hsi_s0:        | dsb 1                              |
| 006B          | 115 | last_stat:     | dsb 1                              |
| 006C          | 116 | pwm_pwr:       | dsb 1                              |
| 006D          | 117 | ios1_bak:      | dsb 1                              |
| 006E          | 118 | TR_COL:        | DSB 1 ; COLLECT TRACE IF TR_COL=00 |
| 006F          | 119 | main_dly:      | dsb 1                              |
|               | 120 |                |                                    |
| 0070          | 121 | max_pwr:       | dsw 1                              |
| 0072          | 122 | max_brk:       | dsw 1                              |
| 0074          | 123 | max_hold:      | dsw 1                              |
| 0076          | 124 | vel_pnt:       | dsw 1                              |
| 0078          | 125 | brk_pnt:       | dsw 1                              |
| 007A          | 126 | pos_pnt:       | dsw 1                              |
| 007C          | 127 | H800_dly:      | dsw 1                              |
| 007E          | 128 | swt1_dly:      | dsw 1                              |
| 0080          | 129 | swt2_dly:      | dsw 1                              |
| 0082          | 130 | min_hsi1:      | dsw 1                              |
| 0084          | 131 | min_hsi1:      | dsw 1                              |
| 0086          | 132 | max_hsi1:      | dsw 1                              |
| 0104          | 133 |                |                                    |
| 0105          | 134 |                |                                    |
| 0100          | 135 | dseg_at:100H   |                                    |

```

0100
0102
0104
008F
008E
0085
0080
001E
001C
001A
0018
0016
0015
0010
0096
009E
009D
009C
2000
2000 0022
2002 1020
2004 0424
2006 8022
2008 1020
200A 2022
200C 1020
200E 1020
002E
2010
2010
2010
2010
2080
2080
2084 B1FF17
2087 1168
2089 A170175C
208D 055C
208F E068FD
2092 88005C
2095 D2F6
2098
2097 B1FF0F
209A B1FF10
003C
136
137 mode_view:    dsb    1
138 count_out:    dsw    1
139 err_view:      dsw    1
140
141
142 $eject
143
144 ; PIN# PORT FLAG USAGE
145
146 ; 22 P1.0 mode0 0 mode1 1 mode2 1 or 0
147 ; 23 P1.1 0 0 1 1
148 ; 24 P1.2 software timer 2 routine enter/leave
149 ; 25 P1.3 Main program toggle
150 ; 26 P1.4 HSI overflow toggle
151 ; 37 P1.5 software timer 0 routine enter/leave
152 ; 38 P1.6 hsi_int enter/leave
153 ; 39 P1.7 software timer 1 routine enter/leave
154 ; 40 P2.6 Input direction (0=reverse, 1=forward)
155 ; 45 P2.7 direction 0=rev, 1=fwd
156
157 cseg at 2000H
158 dcw timer_ovf_int
159 dcw atod_done_int
160 dcw hsi_data_int
161 dcw hso_exec_int
162 dcw hsi_0_int
163 dcw soft_tmr_int
164 dcw ser_port_int
165 dcw external_int
166
167 atod_done_int:
168 hsi_0_int:
169 ser_port_int:
170 external_int:
171
172 cseg at 2080H
173
174 init: ld sp, #0F0H
175 ldb pwm_control, #0FFH
176
177 $EJECT clrb direct
178 ld tmp1, #6000 ; wait about 3 seconds for motor
179 delay: dec tmp1 ; to come to a stop
180 djnz direct, $ ; wait 0.512 milliseconds
181 cmp tmp1, zero
182 jgt $ delay
183
184 ldb port1, #0FFH
185 ldb port2, #0FFH

```



```

209D B12516      186      ldb      IOC1,#00100101B ; Disable HSO. 4,HSO. 5, HSI_INT=first,
353D 31E040      187      382      bccp      t001;0000; Enable PWM,TXD,TIMER1_OVRFLOW_INT
355V 319009      188      384      lpc      t001;000;T.CPU;0005
20A0 71FC0F      189      383      andb     Port1,#11111100B ; clear P1.0,1 (set mode 0)
20A3 B19903      190      385      ldb      HSI_mode,#10011001B ; set hsi.1,3 -; hsi.0,2 +
20A6 B15715      191      387      ldb      IOCO,#01010111B ; Enable all hsi
3554 309003      192      380      lpc      t001;000;0;CPU;000;T2 CLOCK=T2CLK, T2RST=T2RST
3554          193      318      cpl     t001;000;0;CPU;000;T2RST=T2RST
3551 A019FD      194      $eject      dlp      t001;000;0;CPU;000;T2RST=T2RST
3550 85          195      311      bccp     t001;000;0;CPU;000;T2RST=T2RST
20A9 A00400      196      319      ld      zero,hsi_time
20AC 0140        197      312      clr      time
20AE 0142        198      314      clr      time+2
20B0 0128        199      313      clr      timer_2 5550H
20B2 012A        200      315      clr      timer_2+2
20B4 0130        201      317      clr      position
20B6 0132        202      310      clr      position+2
20B8 0154        203      316      clr      last_pos
20BA 0134        204      318      clr      des_pos
20BC 0136        205      311      clr      des_pos+2
20BE 0144        206      319      clr      des_time
20C0 0146        207      322      clr      des_time+2
20C2 A00A56      208      314      ld      last1_time,Timer1
20C5 490008565B  209      323      sub     last2_time,last1_time,#800H
20CA 116D        210      325      clrb    ios1_bak
20CC 1109        211      321      clrb    int_pending
20CE A1F0015E      212      320      ld      out_ptr,#1FOH
20D2 A13C00B2      213      323      ld      min_hsi,#min_hsi_t
20D6 A11E00B4      214      328      ld      min_hsil,#min_hsil_t
20DA A16900B6      215      321      ld      max_hsil,#max_hsil_t
20DE A16E007C      216      329      ld      HSO0_dly,#HSO0_dly_period
20E2 A1FA007E      217      322      ld      swt1_dly,#swt1_dly_period
20E6 A1FA0080      218      324      ld      swt2_dly,#(swt2_dly_period)
20EA A1FF0070      219      323      ld      max_pwr,#max_power
20EE A1FF0072      220      325      ld      max_brk,#max_brake
20F2 A1800074      221      321      ld      max_hold,#maximum_hold
20F6 A180047B      222      320      ld      brk_pnt,#brake_pnt
20FA A164007A      223      323      ld      pos_pnt,#position_pnt
20FE A1100076      224      328      ld      vel_pnt,#velocity_pnt
2102 A1002962      225      321      ld      nxt_pos,#pos_table
2106 B0006C        226      329      ldb     pwm_pwr,zero
2109 B10169        227      322      ldb     pwm_dir,#01h ; FORWARD
          228      324
210C B12D0B        229      323      ldb     int_mask,#00101101B ; Enable tmr_ovf, hsi, swt, HSO, interrupts
210F B13006        230      325      ldb     hso_command,#30H ; set HSO_0
2112 447COA04      231      321      add     hso_time,timer1,HSO0_dly
2116 FD            232      320      nop
2117 FD            233      323      NOP
2118 B13906        234      328      ldb     hso_command,#39H ; set swt_1
211B 447EOA04      235      321      add     hso_time,timer1,swt1_dly
          236      329

```

|               |     |     |   |
|---------------|-----|-----|---|
| 211F FD       |     | 236 | nop   |
| 2120 FD       | 532 | 237 | nop   |
| 2121 B13A06   | 533 | 238 | ldb comhso_command, #3AH                          |
| 2124 44800A04 | 534 | 239 | add hso_time, timer1, swt2_dly                    |
|               | 535 | 240 |   |
| 2128 A00A40   | 536 | 241 | ld timer_time, TIMER1                             |
| 212B A00C2C   | 537 | 242 | ld comtmr2_old, timer2                            |
| 212E FB       | 538 | 243 | eri   |
|               | 539 | 244 |   |
| 212F E7CE06   | 540 | 245 | brw main_prog                                     |
|               | 541 | 246 |   |
|               | 542 | 247 | \$eject   |
|               | 543 | 248 |   |
|               | 544 | 249 |   |
|               | 545 | 250 |   |
|               | 546 | 251 |   |
|               | 547 | 252 |   |
|               | 548 | 253 |   |
| 2200          | 549 | 254 | CSEG AT 2200H                                     |
|               | 550 | 255 |   |
| 2200          | 551 | 256 | timer_ovf_int:                                    |
| 2200 F2       | 552 | 257 | pushf   |
|               | 553 | 258 |   |
| 2201 9016D    | 554 | 259 | orb ios1_bak, IOS1                                |
| 2204 356D05   | 555 | 260 | chk_t1: jbc ios1_bak, 5, tmr_int_done             |
| 2207 0742     | 556 | 261 | inc tmr_time+2                                    |
| 2209 71DF6D   | 557 | 262 | andb ios1_bak, #11011111B ; clear bit 5           |
| 220C          | 558 | 263 | tmr_int_done:                                     |
| 220C F3       | 559 | 264 | popf  |
| 220D F0       | 560 | 265 | ret ; End of timer interrupt routine              |
|               | 561 | 266 |   |
|               | 562 | 267 |   |
|               | 563 | 268 |   |
|               | 564 | 269 |   |
|               | 565 | 270 | SOFTWARE TIMER INTERRUPT SERVICE ROUTINE          |
|               | 566 | 271 |   |
|               | 567 | 272 |   |
| 2220          | 568 | 273 | CSEG AT 2220H                                     |
|               | 569 | 274 |   |
|               | 570 | 275 |   |
| 2220          | 571 | 276 | soft_tmr_int:                                     |
| 2220 F2       | 572 | 277 | pushf   |
| 2221 9016D    | 573 | 278 | orb ios1_bak, IOS1                                |
| 2224          | 574 | 279 | chk_sw0:  |
| 2224 306D03   | 575 | 280 | jbc ios1_bak, 0, chk_sw1                          |
| 2227 71FE6D   | 576 | 281 | andb ios1_bak, #11111110B ; Clear bit 0 - end sw0 |
|               | 577 | 282 | call sw0_expired                                  |
| 222A          | 578 | 283 | chk_sw1:  |
| 222A 316D06   | 579 | 284 | jbc ios1_bak, 1, chk_sw2                          |
| 222D 71FD6D   | 580 | 285 | andb ios1_bak, #11111101B ; Clear bit 1           |
|               | 581 |     |   |

```

2230 EFCD03      286      call      swt1_expired
2233      287      chk_sw2:      andb      ios1_bak,2,chk_sw23
2233 326D06      288      andb      ios1_bak,#11111011B      ; Clear bit 2
2236 71F86D      289      call      swt2_expired
2239 EF4401      290      chk_sw3:      jbc      ios1_bak,4,swt_int_done
223C      291      andb      ios1_bak,#11110111B      ; Clear bit 3
223C 346D03      292      call      swt3_expired
223F 71F76D      293      swt_int_done:      popf
2242      294      ret      ; END OF SOFTWARE TIMER INTERRUPT ROUTINE
2242 F3          295
2243 F0          296
2243 F0          297
2243 F0          298
2243 F0          299
2243 F0          300
2243 F0          301
2243 F0          302
2243 F0          303
2243 F0          304
2243 F0          305
2243 F0          306
2243 F0          307
2243 F0          308
2243 F0          309
2243 F0          310
2243 F0          311
2243 F0          312
2243 F0          313
2243 F0          314
2243 F0          315
2243 F0          316
2243 F0          317
2243 F0          318
2243 F0          319
2243 F0          320
2243 F0          321
2243 F0          322
2243 F0          323
2243 F0          324
2243 F0          325
2243 F0          326
2243 F0          327
2243 F0          328
2243 F0          329
2243 F0          330
2243 F0          331
2243 F0          332
2243 F0          333
2243 F0          334
2243 F0          335

```

```

22B3 643C30      236      in_fwd: add      position,delta_p
22B6 A40032      237      addc     position+2,zero
22B9 2006        238      br       chk_mode
239
22BB 683C30      239      in_rev: sub      position,delta_p
22BE AB0032      241      subc     position+2,zero
242
22C1            243      chk_mode:
22C1 4866285C      244      sub      tmp1,Timer_2,old_t2      ; Check count difference in tmp1
22C5 8705005C      245      cmp      tmp1,#5                ; set model1 if count is too low
22C9 D21C        246      jgt      end_swt0            ; count <= 5 HSI
247
22CB            248      set_model:
22CB 71FD0F      249      andb     Port1,#1111101B          ; Clear P1.1, set P1.0 (set mode 1)
22CE 91010F      250      orb      Port1,#00000001B
22D1 B10515      251      ldb      IDCO,#00000101B        ; enable HSI 0 and 1
22D4 A00400      252      ld       zero, HSI_TIME
22D7 48840A56      253      sub      last1_time,Timer1,min_hsi1
254
22E8            255      $EJECT
22E8 700C8B      256
22DB            257      clr_hsi:
22DB A00400      258      ld       ZERO, HSI_TIME
22DE 717F6D      259      andb     ios1_bak,#01111111B      ; clear bit 7
22E1 90166D      260      orb      ios1_bak,ios1
22E4 3F6DF4      261      jbs      ios1_bak,7,clr_hsi      ; If hsi is triggered then clear hsi
262
22E7            263      end_swt0:
22E7 A02866      264      ld       old_t2,TIMER_2
22EA 71DF0F      265      andb     port1,#11011111B          ; clear P1.5
22ED F3          266      POPF
22EE F0          267      ret
268
269
270
271
272      SOFTWARE TIMER ROUTINE 2
273
274
275      CSEQ AT 2380H
276
277      swt2_expired:
278      pushf
279      ldb      hso_command,#3AH          ; set swt_2
280      add      hso_time,timer1,swt2_dly
281
282      orb      port1,#00000100B          ; set port 1.2
283      cmp      out_ptr,#7ffH
284      bnh      pulsing
285      ld       out_ptr,#1f0H

```



```

2395      386
2395 304E0C      387      pulsing:
2395      388          jbc      tr_col,0,swt2_done
2395      389
239B C25F32      390          st      position+2,[out_ptr]+      ; position high, position low
239B C25F30      391          st      position,[out_ptr]+
2395      392
239E C25F68      393          st      direct,[out_ptr]+
23A1 C25F6C      394          st      pwm_pwr,[out_ptr]+
2395      395
2395      396          ; store 8 bytes externally
2395      397
23A4      398      swt2_done:
23A4 48560A5C      399          sub      tmp1,timer1,last1_time
23A8 8900185C      400          cmp      tmp1,#1800H
23AC D104      401          jnh      swt2_ret      ; keep (Timer1-last1_time)<2000H
2395      402
23AE 65001056      403          add      last1_time,#1000H
23B2      404      swt2_ret:
23B2 71FB0F      405          andb     port1,#11111011B      ; clear port1.2
23B5 F3      406          popf
23B6 F0      407          ret
2395      408
2395      409      $EJECT
2395      410      ;
2395      411      ; HSI DATA AVAILABLE INTERRUPT ROUTINE
2395      412      ;
2395      413      ;
2395      414      ; This routine keeps track of the current time and position of the motor.
2395      415      ; The upper word of information is provided by the timer overflow routine.
2395      416
2395      417      CSEG AT 2400H
2400      418      now_mode_1: br      in_mode_1      ; used to save execution time for
2402 20C7      419      no_int1: br      no_int      ; worst case loop
2395      420
2404 F2      421      hsi_data_int: pushf
2405 91400F      422          orb      port1,#01000000B      ; set P1.6
2408 717F6D      423          andb     ios1_bak,#01111111B      ; Clear ios1_bak.7
240B 90166D      424          orb      ios1_bak,ios1
240E 376DF1      425          jbc      ios1_bak,7,no_int1      ; If hsi is not triggered then
2395      426          ; jump to no_int
2411      427      get_values:
2411 A00C28      428          ld      timer_2,TIMER2
2414 5155066A      429          andb     hsi_s0,HSI_STATUS,#01010101B
2418 A00440      430          ld      time,HSI_TIME
2395      431
241B 380FE2      432          jbs      port1,0,now_mode_1      ; jump if in mode 1
2395      433
241E      434      In_mode_0:
241E 386A0B      435          jbs      hsi_s0,0,a_rise

```

```

2421 3A6A2C          436          jbs      hsi_s0,2,a_fall
2424 3C6A4D          437          jbs      hsi_s0,4,b_rise
2427 3E6A5A          438          jbs      hsi_s0,6,b_fall
242A 2094            439          br       no_cnt
2440
242C A0565B          441      a_rise: ld      last2_time,last1_time
242F A04056          442          ld      last1_time,time
2432 685B40          443          sub      time,last2_time
2435 88B240          444          cmp      time,min_hsi
243B D906            445          jh       tst_statr
2440
243A 91010F          446      ;set model-
243D B10515          447          orb      Port1,#00000001B      ; Set P1.0 (in mode 1)
2440                448          ldb      IOCO,#00000101B      ; Enable HSI 0 and 1
2440                449      tst_statr:
2440 3E6B5B          450          jbs      last_stat,6,going_fwd
2443 3C6B67          451          jbs      last_stat,4,going_rev
2446 3A6B50          452          jbs      last_stat,2,change_dir
2449 98006B          453          cmpb     last_stat,zero
244C DF46            454          je       first_time      ; first time in mode0
244E 27B2            455          br       no_int1
2456
2450 A0565B          456      a_fall: ld      last2_time,last1_time
2453 A04056          457          ld      last1_time,time
2456 685B40          458          sub      time,last2_time
2459 88B240          459          cmp      time,min_hsi
245C D906            460          jh       tst_statf
2456
245E 91010F          461          orb      Port1,#00000001B      ; Set P1.0 (in mode 1)
2461 B10515          462          ldb      IOCO,#00000101B      ; Enable HSI 0 and 1
2466
245E 91010F          463          orb      Port1,#00000001B      ; Set P1.0 (in mode 1)
2461 B10515          464          ldb      IOCO,#00000101B      ; Enable HSI 0 and 1
2466
2464                465      $EJECT
2464                466      tst_statf:
2464 3C6B37          467          jbs      last_stat,4,going_fwd
2467 3E6B43          468          jbs      last_stat,6,going_rev
246A 386B2C          469          jbs      last_stat,0,change_dir
246D 98006B          470          cmpb     last_stat,zero
2470 DF22            471          je       first_time      ; first time in mode0
2472 2057            472          br       no_int
2473
2474 386B27          473      b_rise: jbs      last_stat,0,going_fwd
2477 3A6B33          474          jbs      last_stat,2,going_rev
247A 3E6B1C          475          jbs      last_stat,6,change_dir
247D 98006B          476          cmpb     last_stat,zero
2480 DF12            477          je       first_time      ; first time in mode0
2482 2047            478          br       no_int
2480
2484 3A6B17          479      b_fall: jbs      last_stat,2,going_fwd
2487 386B23          480          jbs      last_stat,0,going_rev
248A 3C6B0C          481          jbs      last_stat,4,change_dir
248D 98006B          482          cmpb     last_stat,zero
2490 DF02            483          je       first_time      ; first time in mode0

```

```

2492 2037 1104          486          br      no_int
2493 812A0F          487          ;
2494          488      first_time:
2494 C46B6A          489          stb      hsi_s0,last_stat
2497 2072          490          br      done_chk      ; add delta position
2497 81800E          491          ;
2498 81800E          492          ;
2499          493      change_dir:
2499 1268          494          andb     notb      direct
249B 30680F          495          no_inc: jbc      direct,0,going_rev
249C          496          ;
249E          497      going_fwd:
249E 914010          498          orb      PORT2,#01000000B      ; set P2.6
24A1 B10168          499          ldb      direct,#01      ; direction = forward
24A4 65010030        500          add      position,#01
24AB A40032          501          andb     position+2,zero
24AB 200D          502          br      st_stat
24AD 514F          503      going_rev:
24AD 71BF10          504          andb     PORT2,#10111111B      ; clear P2.6
24B0 B10068          505          ldb      direct,#00      ; direction = reverse
24B3 69010030        506          sub      position,#01
24B7 AB0032          507          subc     position+2,zero
24B8 690032          508          ;
24BA 690030          509          st_stat:
24BA C46B6A          510          andb     hsi_s0,last_stat
24BD 30680F          511          load_last:
24BD A0282C          512          ld      tmr2_old,timer_2
24C0 717F6D          513          no_cnt: andb     ios1_bak,#01111111B      ; clr bit 7
24C3 90166D          514          orb      ios1_bak,ios1
24C6 376D02          515          jbc      ios1_bak,7,no_int
24C9 2746          516          again: br      get_values
24CB 71BF0F          517          ;
24CE F3          518          no_int: andb     port1,#10111111B      ; Clear P1.6
24CF F0          519          popf
24D0          520          ret
24D0          521          ; end of hsi_data interrupt routine
24D0          522          ; Routine for mode 1 follows and then returns to "load_last"
24D0          523          ;
24D0          524          In_mode_1:      ; mode 1 HSI routine
24D0          525          ;
24D0 51506A5C        526          ;
24D0 D7EA          527          andb     tmp1,hsi_s0,#01010000B      ; If port 1 is disabled then clear port
24D4          528          jne     no_cnt
24D6          529          cmp_time:
24D6          530          ; Procedure which sets mode 1 also
24D6          531          ; sets times to pass the tests
24D7 A05658          532          ld      last2_time,last1_time
24D7 A04056          533          ld      last1_time,time
24DC 485B405C        534          sub      tmp1,time,last2_time
24E0 88B45C          535          cmp      tmp1,min_hsil

```

```

24E3 D914 2C      536          jh b      check_max_time
24E4 4838 02C      537          sub     #0, timer_2
24E5              538      set_mode_2:
24E5 9102 0F      539          orb     Port1, #000000010B      ; Set P1.1 (in mode 2)
24E8 B100 15      540          ldb     IOCO, #000000000B      ; Disable all HSI
24E8 A004 00      541      mt_hsi: ld     zero, hsi_time      ; empty the hsi fifo
24EE 717F 6D      542          andb    ios1_bak, #01111111B      ; clear bit 7
24F1 9016 6D      543          orb     ios1_bak, ios1
24F4 3F6D F4 2C    544          jbs     ios1_bak, 7, mt_hsi      ; If hsi is triggered then clear hsi
24F7 2012          545          br      done_chk
24F7              546
24F9              547      check_max_time:
24F9 4858 405C      548          sub     tmp1, time, last2_time
24FD 8886 5C      549          cmp     tmp1, max_hsi1      ; max_hsi = addition to min_hsi for
24FD              550          ; total time
2500 D109          551          jnh     done_chk
2500              552
2502              553      set_mode_0:
2502 71FC 0F      554          andb    Port1, #11111100B      ; clear P1.0,1 set mode 0)
2505 B155 15      555          ldb     IOCO, #01010101B      ; Enable all HSI
2508 B000 6B      556          ldb     last_stat, zero
2508              557
2508              558      done_chk:
2508 482C 283C      559          sub     delta_p, timer_2, tmr2_old      ; get timer2 count difference
250F 3068 08      560          jbc     direct, 0, add_rev
2512 643C 30      561      add_fwd: add     position, delta_p
2512 643C 30      562          add     position+2, zero
2515 A400 32      563          addc    load_last
2518 27A3          564          br      add_rev
251A 683C 30      565          sub     position, delta_p
251A 683C 30      566          subc    position+2, zero
251D A800 32      567          br      load_last
2520 279B          568          load_last
2520              569
2520              570      $reject
2520              571      ;
2520              572      ;
2520              573      ; SOFTWARE TIMER ROUTINE 1
2520              574      ;
2520              575      CSEG AT 2600H
2520              576
2520              577      swt1_expired:
2520              578
2520              579          pushf
2520 F2          580          orb     port1, #10000000B      ; set port1.7
2520 9180 0F      581          ldb     int_mask, #00001101B      ; enable HSI, T0vf, HSD
2520 B10D 08      582
2520              583
2520              584          ldb     HSO_COMMAND, #39H
2520 B139 06      585          add     HSO_TIME, TIMER1, swt1_dly
2520 A47E 0A04      585

```



```

260E A0464A      586
2611 A0363A      587      ld      time_err+2,des_time+2      ; Calculate time & position error
2614 48404448     588      ld      pos_err+2,des_pos+2
2618 A8424A      589      sub     time_err,des_time,time      ; values are set
261B 48303438     590      sub     time_err+2,time+2
261F A8323A      591      sub     pos_err,des_pos,position
2622 FB          592      sub     pos_err+2,position+2
2623 48484C52     593      $EJECT
2627 A0484C      594      EI
2628          595
2629          596
262A 48384E50     597      sub     time_delta,last_time_err,time_err
262E A0384E      598      ld      last_time_err,time_err
262F          599
2630          600      sub     pos_delta,last_pos_err,pos_err
2631          601      ld      last_pos_err,pos_err
2632          602      ; Time_err = Desired time to finish - current time
2633          603      ; Pos_err = Desired position to finish - current position
2634          604      ; Pos_delta = Last position error - Current position error
2635          605      ; Time_delta = Last time error - Current time error
2636          606      ; note that errors should get smaller so deltas will be
2637          607      ; positive for forward motion (time is always forward)
2638          608
2639          609
263A          610      chk_dir:      ;
263B          611      cmp     pos_err+2,zero
263C          612      jge     go_forward
263D          613
263E          614      go_backward:
263F          615      neg     pos_err      ; Pos_err = ABS VAL (pos_err)
2640          616      ldb     pwm_dir,#00h
2641          617      cmp     pos_err+2,#0ffffH
2642          618      jne     ld_max
2643          619      chk_brk
2644          620
2645          621      go_forward:
2646          622      ldb     pwm_dir,#01H
2647          623      cmp     pos_err+2,zero
2648          624      jge     chk_brk
2649          625      $EJECT
264A          626
264B          627      ld_max:      ldb     pwm_pwr,max_pwr
264C          628      br     chk_sanity
264D          629
264E          630      Chk_brk:
264F          631      cmp     pos_err,pos_pnt
2650          632      jnh     hold_position      ; position_error < position_control_point
2651          633      cmp     pos_err,brk_pnt
2652          634
2653          635
2654          636
2655          637
2656          638
2657          639
2658          640
2659          641
2660          642
2661          643
2662          644
2663          645
2664          646
2665          647
2666          648
2667          649
2668          650
2669          651
2670          652
2671          653
2672          654
2673          655
2674          656
2675          657
2676          658
2677          659
2678          660
2679          661
2680          662
2681          663
2682          664
2683          665
2684          666
2685          667
2686          668
2687          669
2688          670
2689          671
2690          672
2691          673
2692          674
2693          675
2694          676
2695          677
2696          678
2697          679
2698          680
2699          681
2700          682
2701          683
2702          684
2703          685
2704          686
2705          687
2706          688
2707          689
2708          690
2709          691
2710          692
2711          693
2712          694
2713          695
2714          696
2715          697
2716          698
2717          699
2718          700
2719          701
2720          702
2721          703
2722          704
2723          705
2724          706
2725          707
2726          708
2727          709
2728          710
2729          711
2730          712
2731          713
2732          714
2733          715
2734          716
2735          717
2736          718
2737          719
2738          720
2739          721
2740          722
2741          723
2742          724
2743          725
2744          726
2745          727
2746          728
2747          729
2748          730
2749          731
2750          732
2751          733
2752          734
2753          735
2754          736
2755          737
2756          738
2757          739
2758          740
2759          741
2760          742
2761          743
2762          744
2763          745
2764          746
2765          747
2766          748
2767          749
2768          750
2769          751
2770          752
2771          753
2772          754
2773          755
2774          756
2775          757
2776          758
2777          759
2778          760
2779          761
2780          762
2781          763
2782          764
2783          765
2784          766
2785          767
2786          768
2787          769
2788          770
2789          771
2790          772
2791          773
2792          774
2793          775
2794          776
2795          777
2796          778
2797          779
2798          780
2799          781
2800          782
2801          783
2802          784
2803          785
2804          786
2805          787
2806          788
2807          789
2808          790
2809          791
2810          792
2811          793
2812          794
2813          795
2814          796
2815          797
2816          798
2817          799
2818          800
2819          801
2820          802
2821          803
2822          804
2823          805
2824          806
2825          807
2826          808
2827          809
2828          810
2829          811
2830          812
2831          813
2832          814
2833          815
2834          816
2835          817
2836          818
2837          819
2838          820
2839          821
2840          822
2841          823
2842          824
2843          825
2844          826
2845          827
2846          828
2847          829
2848          830
2849          831
2850          832
2851          833
2852          834
2853          835
2854          836
2855          837
2856          838
2857          839
2858          840
2859          841
2860          842
2861          843
2862          844
2863          845
2864          846
2865          847
2866          848
2867          849
2868          850
2869          851
2870          852
2871          853
2872          854
2873          855
2874          856
2875          857
2876          858
2877          859
2878          860
2879          861
2880          862
2881          863
2882          864
2883          865
2884          866
2885          867
2886          868
2887          869
2888          870
2889          871
2890          872
2891          873
2892          874
2893          875
2894          876
2895          877
2896          878
2897          879
2898          880
2899          881
2900          882
2901          883
2902          884
2903          885
2904          886
2905          887
2906          888
2907          889
2908          890
2909          891
2910          892
2911          893
2912          894
2913          895
2914          896
2915          897
2916          898
2917          899
2918          900
2919          901
2920          902
2921          903
2922          904
2923          905
2924          906
2925          907
2926          908
2927          909
2928          910
2929          911
2930          912
2931          913
2932          914
2933          915
2934          916
2935          917
2936          918
2937          919
2938          920
2939          921
2940          922
2941          923
2942          924
2943          925
2944          926
2945          927
2946          928
2947          929
2948          930
2949          931
2950          932
2951          933
2952          934
2953          935
2954          936
2955          937
2956          938
2957          939
2958          940
2959          941
2960          942
2961          943
2962          944
2963          945
2964          946
2965          947
2966          948
2967          949
2968          950
2969          951
2970          952
2971          953
2972          954
2973          955
2974          956
2975          957
2976          958
2977          959
2978          960
2979          961
2980          962
2981          963
2982          964
2983          965
2984          966
2985          967
2986          968
2987          969
2988          970
2989          971
2990          972
2991          973
2992          974
2993          975
2994          976
2995          977
2996          978
2997          979
2998          980
2999          981
3000          982
3001          983
3002          984
3003          985
3004          986
3005          987
3006          988
3007          989
3008          990
3009          991
3010          992
3011          993
3012          994
3013          995
3014          996
3015          997
3016          998
3017          999
3018          1000

```

```

265B D9F1          634          jh          ld_max          ; position_error > brake_point
265A              635
265A 880050        636          braking:
265D D602          637          cmp          pos_delta, zero
265F 0350          638          jge          chk_delta
2661              639          neg          pos_delta
2661 887650        640          chk_delta:
2664 D10D          641          cmp          pos_delta, vel_pnt ; velocity = pos_delta/sample_time
2664 B0726C        642          jnh          hold_position ; jmp if ABS(velocity) < vel_pnt
2666 B0726C        643
2669 B06824        644          brake: ldb          pwm_pwr, max_brk
266C 1224          645          ldb          tmp, direct ; If braking apply power in opposite
266E B02469        646          notb         ; direction of current motion
2671 2030          647          ldb          pwm_dir, tmp
2673              648
2673 8902003B       649          br          ld_pwr
2677 D906          650
2679 0126          651          Hold_position:
267B 015A          652          cmp          pos_err, #02 ; position hold mode
267D 201F          653          jh          calc_out ; if position error < 2 then turn off power
267F 5DFF7424      654          clr          tmp+2
2683 6C3824        655          boost
2686 880050        656          output
2689 D709          657
268B 6504005A      658          calc_out:
268F 645A26        659          mulub         tmp, max_hold, #255
2692 2002          660          mulu          tmp, pos_err ; Tmp = pos_err * max_hold
2694 015A          661          cmp          pos_delta, zero
2696 887426        662          jne          no_bst
2699 D103          663          add          boost, #04 ; Boost is integral control
269E B0266C        664          add          tmp+2, boost ; TMP+2 = MSB(pos_err*max_hold)
26A1 2000          665          br          ck_max
26A3 B06C64        666          no_bst: clr
26A6 1264          667          ck_max: cmp          tmp+2, max_hold
26A8 38690A        668          jnh          output
26AE B0266C        669          maxed: ld          tmp+2, max_hold
26B1              670          output: ldb          pwm_pwr, tmp+2
26B3              671
26B5              672
26B7              673          chk_sanity:
26B9              674          brp          ld_pwr
26BB              675          ;;
26BD              676          ;;
26BF              677          $EJECT
26C1              678
26C3              679          ld_pwr:
26C5              680          ldb          rpwr, pwm_pwr
26C7              681          notb
26C9              682          jbs          pwm_dir, 0, p2fwd
26CB              683          jbs

```

```

26AB FA
26AC 717F10
26AF B06417
26B2 FB
26B3 2008
26B5 FA
26B6 918010
26B9 B06417
26BC FB

26BD
26BD 88004A
26C0 D225
26C2 89202962
26C6 DE06
26C8 A1002962
26CC 0142
26CE 00000000
26CE A26334
26D1 A26336
26D4 A26346
26D7 A26370
26DA A07072
26DD 646034
26E0 A40036
26E3 4830344E
26E7 717F0F
26EA F3
26EB F0
2800 620806
2800 90166D
2803 366D09
2806 71BF6D
2809 95100F
280C EFF5FB

684 p2bkwd: DI
685 andb port2, #01111111B ; clear P2.7
686 ldb pwm_control, rpwr
687 EI
688 br pwrset
689 p2fwd: DI
690 orb port2, #10000000B ; set P2.7
691 ldb pwm_control, rpwr
692 EI
693
694 pwrset:
695 cmp time_err+2, zero ; do pos_table when err is negative
696 jgt end_p
697 br end_p
698
699 cmp nxt_pos, #(32+pos_table)
700 jlt get_vals ; jump if lower
701 ld nxt_pos, #pos_table
702 clr time+2
703 get_vals:
704 ld des_pos, [nxt_pos]+
705 ld des_pos+2, [nxt_pos]+
706 ld des_time+2, [nxt_pos]+
707 ld max_pwr, [nxt_pos]+
708 ld max_brk, max_pwr
709 add des_pos, offset
710 addc des_pos+2, zero
711 sub last_pos_err, des_pos, position
712
713 end_p: andb port1, #01111111B ; clear P1.7
714
715 popf
716 ret
717
718 $EJECT
719
720
721
722
723
724
725
726 CSEG at 2800H
727
728 MAIN_PROG:
729 orb ios1_bak, ios1
730 jbc ios1_bak, 6, control
731 andb ios1_bak, #10111111B ; clear ios1_bak.6
732 xorb Port1, #00010000B ; Compl Bit P1.4
733 call HSI_DATA_INT ; prevent lockup

```

```

280F ELL2LE      734 control: 011
280F 912D08      735         orb     int_mask,#00101101B ; enable hsi, hso, sut, tovf interrupts
2812 FD EVD      736         nop
2813 FD 00A      737         nop
2814 FD 00D      738         nop
2815 E06FFD      739         djnz    main_dly,$
2818 FD          740         nop
2819 95080F      741         xorb    port1,#00001000B ; compliment p1.3
281C 27E2        742         BR      MAIN_PROG
                743
                744
2900            745 CSEG AT 2900H
                746
2900            747 pos_table:
                748
2900 00000000      749         dcl     00000000H ; position 0
2904 20008000      750         dcw     0020H, 0080H ; next time, power
2908 00C00000      751         dcl     0000C000H ; position 1
290C 40004000      752         dcw     0040H, 0040H ; next time, power
2910 00000000      753         dcl     00000000H ; position 2
2914 6000C000      754         dcw     0060H, 00C0H ; next time, power
2918 0080FFFF      755         dcl     0080FFFFH ; position 3
291C 80008000      756         dcw     0080H, 0080H ; next time, power
                757
2920 00080000      758         dcl     00000800H ; position 4
2924 58008000      759         dcw     0058H, 0080H ; next time, power
2928 00300000      760         dcl     00003000H ; position 5
292C 7000FF00      761         dcw     0070H, 00FFH ; next time, power
2930 00000000      762         dcl     00000000H ; position 6
2934 9000F000      763         dcw     0090H, 00F0H ; next time, power
2938 00000000      764         dcl     00000000H ; position 7
293C 9100F000      765         dcw     0091H, 00F0H ; next time, power
                766
2940 80303030      767         END
                768
ASSEMBLY COMPLETED, NO ERROR(S) FOUND.

```







## 7.0 INTRODUCTION

The MCS<sup>®</sup>-51 family of 8-bit microcontrollers consists of the devices listed in Table 1, all of which are based on the MCS-51 architecture shown in Figure 7-1. The original 8051 was built in HMOS I technology. The HMOS II version, which is the device currently in production, is called the 8051AH. The term "8051", however, is still

often used to generically refer to all of the MCS-51 family members. This is the case throughout this manual, except where specifically stated otherwise. Also for brevity, the term "8052" is used to refer to both the 8052 and the 8032, unless otherwise noted.

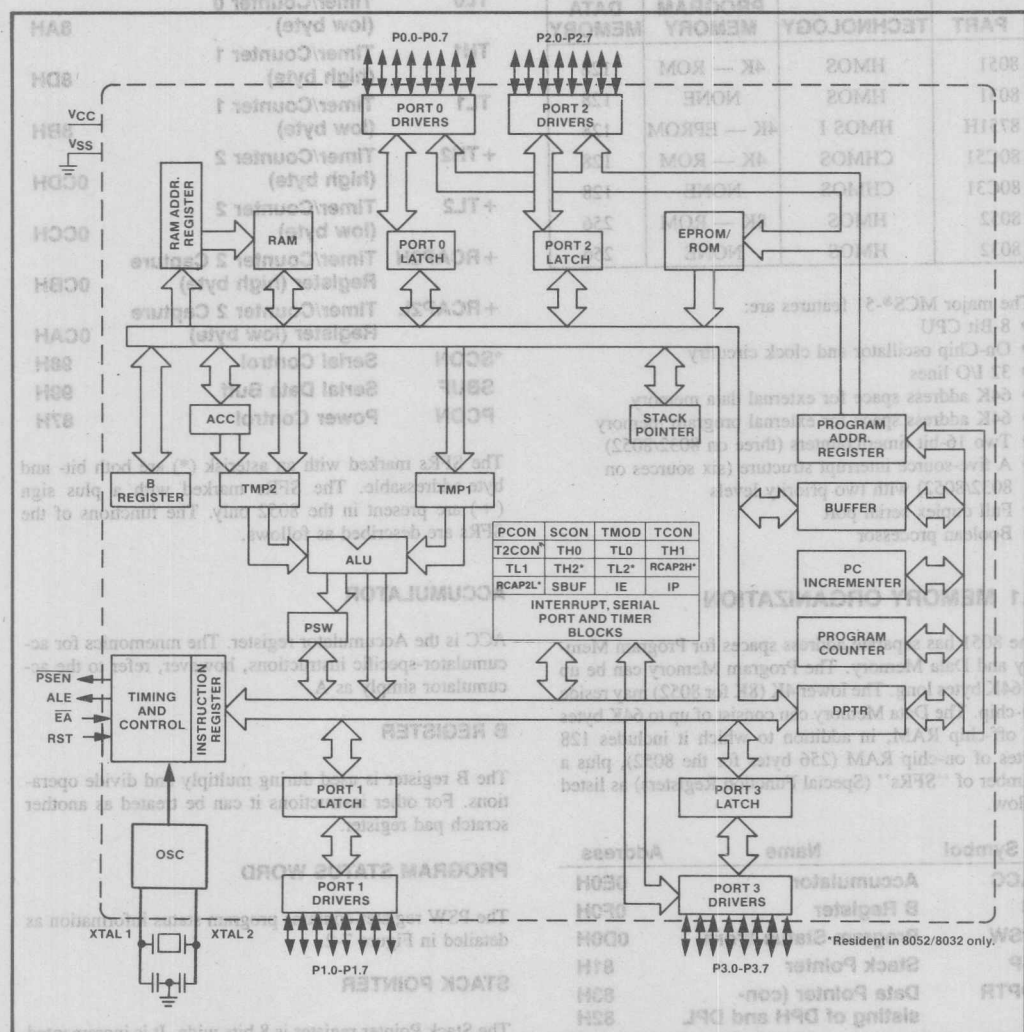


Figure 7-1. MCS-51 Architectural Block Diagram

## MCS<sup>®</sup>-51 ARCHITECTURE

The newest MCS-51 members, the 8032 and 8052, have more on-chip memory and an additional 16-bit timer/counter. The new timer can be used as a timer, a counter, or to generate baud rates for the serial port. As a timer/counter, it operates in either a 16-bit auto-reload mode or a 16-bit "capture" mode. This new feature is described in Section 7.6.2.

Pinouts are shown in the individual data sheets and on the inside back cover of this handbook.

Table 1. MCS<sup>®</sup>-51 Family Members

| PART  | TECHNOLOGY | ON-CHIP PROGRAM MEMORY | ON-CHIP DATA MEMORY |
|-------|------------|------------------------|---------------------|
| 8051  | HMOS       | 4K — ROM               | 128                 |
| 8031  | HMOS       | NONE                   | 128                 |
| 8751H | HMOS I     | 4K — EPROM             | 128                 |
| 80C51 | CHMOS      | 4K — ROM               | 128                 |
| 80C31 | CHMOS      | NONE                   | 128                 |
| 8052  | HMOS       | 8K — ROM               | 256                 |
| 8032  | HMOS       | NONE                   | 256                 |

The major MCS<sup>®</sup>-51 features are:

- 8-Bit CPU
- On-Chip oscillator and clock circuitry
- 32 I/O lines
- 64K address space for external data memory
- 64K address space for external program memory
- Two 16-bit timer/counters (three on 8032/8052)
- A five-source interrupt structure (six sources on 8032/8052) with two priority levels
- Full duplex serial port
- Boolean processor

### 7.1 MEMORY ORGANIZATION

The 8051 has separate address spaces for Program Memory and Data Memory. The Program Memory can be up to 64K bytes long. The lower 4K (8K for 8052) may reside on-chip. The Data Memory can consist of up to 64K bytes of off-chip RAM, in addition to which it includes 128 bytes of on-chip RAM (256 bytes for the 8052), plus a number of "SFRs" (Special Function Registers) as listed below.

| Symbol | Name                                     | Address    |
|--------|--|------------|
| *ACC   | Accumulator                              | 0E0H       |
| *B     | B Register                               | 0F0H       |
| *PSW   | Program Status Word                      | 0D0H       |
| SP     | Stack Pointer                            | 81H        |
| DPTR   | Data Pointer (consisting of DPH and DPL) | 83H<br>82H |
| *P0    | Port 0                                   | 80H        |
| *P1    | Port 1                                   | 90H        |

| Symbol   | Name   | Address |
|----------|--|---------|
| *P2      | Port 2                                       | 0A0H    |
| *P3      | Port 3                                       | 0B0H    |
| *IP      | Interrupt Priority Control                   | 0B8H    |
| *IE      | Interrupt Enable Control                     | 0A8H    |
| TMOD     | Timer/Counter Mode Control                   | 89H     |
| *TCON    | Timer/Counter Control                        | 88H     |
| + *T2CON | Timer/Counter 2 Control                      | 0C8H    |
| TH0      | Timer/Counter 0 (high byte)                  | 8CH     |
| TL0      | Timer/Counter 0 (low byte)                   | 8AH     |
| TH1      | Timer/Counter 1 (high byte)                  | 8DH     |
| TL1      | Timer/Counter 1 (low byte)                   | 8BH     |
| + TH2    | Timer/Counter 2 (high byte)                  | 0CDH    |
| + TL2    | Timer/Counter 2 (low byte)                   | 0CCH    |
| + RCAP2H | Timer/Counter 2 Capture Register (high byte) | 0CBH    |
| + RCAP2L | Timer/Counter 2 Capture Register (low byte)  | 0CAH    |
| *SCON    | Serial Control                               | 98H     |
| SBUF     | Serial Data Buff                             | 99H     |
| PCON     | Power Control                                | 87H     |

The SFRs marked with an asterisk (\*) are both bit- and byte-addressable. The SFRs marked with a plus sign (+) are present in the 8052 only. The functions of the SFRs are described as follows.

### ACCUMULATOR

ACC is the Accumulator register. The mnemonics for accumulator-specific instructions, however, refer to the accumulator simply as A.

### B REGISTER

The B register is used during multiply and divide operations. For other instructions it can be treated as another scratch pad register.

### PROGRAM STATUS WORD

The PSW register contains program status information as detailed in Figure 7-2.

### STACK POINTER

The Stack Pointer register is 8 bits wide. It is incremented before data is stored during PUSH and CALL executions. While the stack may reside anywhere in on-chip RAM,



the Stack Pointer is initialized to 07H after a reset. This causes the stack to begin at location 08H.

## DATA POINTER

The Data Pointer (DPTR) consists of a high byte (DPH) and a low byte (DPL). Its intended function is to hold a 16-bit address. It may be manipulated as a 16-bit register or as two independent 8-bit registers.

## PORTS 0 to 3

P0, P1, P2 and P3 are the SFR latches of Ports 0, 1, 2 and 3, respectively.

## SERIAL DATA BUFFER

The Serial Data Buffer is actually two separate registers, a transmit buffer and a receive buffer register. When data is moved to SBUF, it goes to the transmit buffer where it is held for serial transmission. (Moving a byte to SBUF is what initiates the transmission.) When data is moved from SBUF, it comes from the receive buffer.

## TIMER REGISTERS

Register pairs (TH0, TL0), (TH1, TL1), and (TH2, TL2) are the 16-bit counting registers for Timer/Counters 0, 1, and 2, respectively.

## CAPTURE REGISTERS

The register pair (RCAP2H, RCAP2L) are the capture registers for the Timer 2 "capture mode." In this mode, in response to a transition at the 8052's T2EX pin, TH2 and TL2 are copied into RCAP2H and RCAP2L. Timer

2 also has a 16-bit auto-reload mode, and RCAP2H and RCAP2L hold the reload value for this mode. More about Timer 2's features in Section 7.6.2.

## CONTROL REGISTERS

Special Function Registers IP, IE, TMOD, TCON, T2CON, SCON, and PCON contain control and status bits for the interrupt system, the timer/counters, and the serial port. They are described in later sections.

## 7.2 OSCILLATOR AND CLOCK CIRCUIT

XTAL1 and XTAL2 are the input and output of a single-stage on-chip inverter, which can be configured with off-chip components as a Pierce oscillator, as shown in Figure 7-3. The on-chip circuitry, and selection of off-chip components to configure the oscillator are discussed in Section

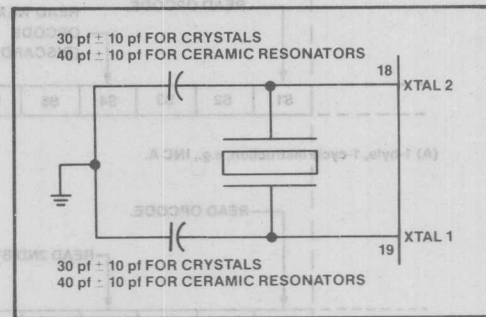


Figure 7-3. Crystal/Ceramic Resonator Oscillator

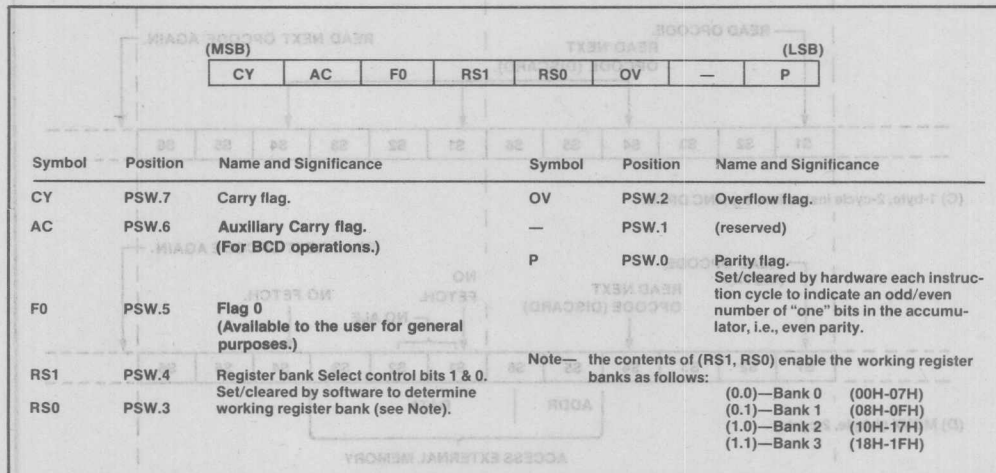


Figure 7-2. PSW: Program Status Word Register

## MCS®-51 ARCHITECTURE

7.13. A more detailed discussion will be found in Application Note AP-155, "Oscillators for Microcontrollers," which is included in this manual.

The oscillator, in any case, drives the internal clock generator. The clock generator provides the internal clocking signals to the chip. The internal clocking signals are at half the oscillator frequency, and define the internal

phases, states, and machine cycles, which are described in the next section.

### 7.3 CPU TIMING

A machine cycle consists of 6 states (12 oscillator periods). Each state is divided into a Phase 1 half, during which the Phase 1 clock is active, and a Phase 2 half,

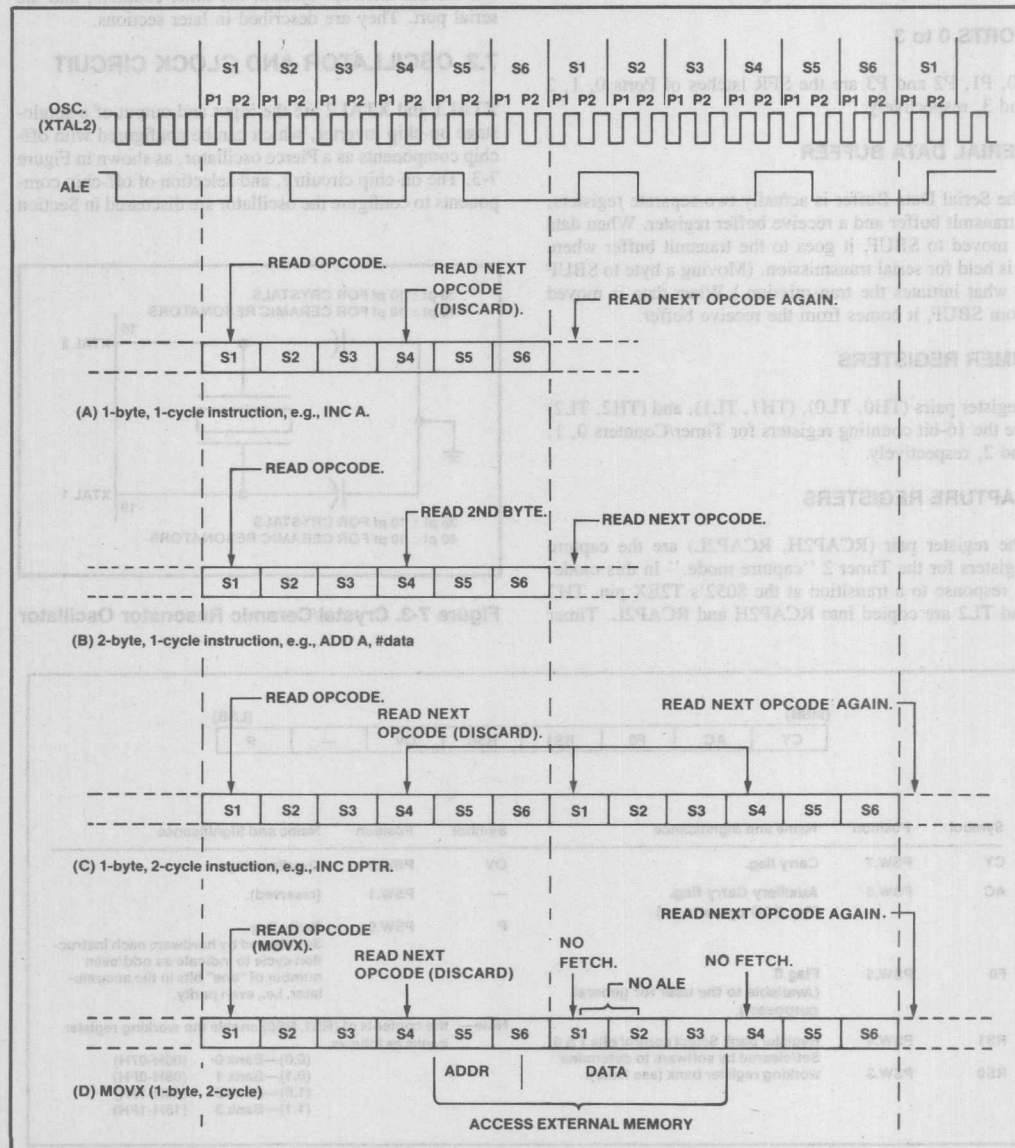


Figure 7-4. 8051 Fetch/Execute Sequences

during which the Phase 2 clock is active. Thus, a machine cycle consists of 12 oscillator periods, numbered S1P1 (State 1, Phase 1), through S6P2 (State 6, Phase 2). Each phase lasts for one oscillator period. Each state lasts for two oscillator periods. Typically, arithmetic and logical operations take place during Phase 1 and internal register-to-register transfers take place during Phase 2.

The diagrams in Figure 7-4 show the fetch/execute timing referenced to the internal states and phases. Since these internal clock signals are not user accessible, the XTAL2 oscillator signal and the ALE (Address Latch Enable) signal are shown for external reference. ALE is normally activated twice during each machine cycle: once during S1P2 and S2P1, and again during S4P2 and S5P1.

Execution of a one-cycle instruction begins at S1P2, when the opcode is latched into the Instruction Register. If it is a two-byte instruction, the second byte is read during S4 of the same machine cycle. If it is a one-byte instruction, there is still a fetch at S4, but the byte read (which would be the next opcode), is ignored, and the Program Counter is not incremented. In any case, execution is complete at

the end of S6P2. Figures 7-4A and 7-4B show the timing for a 1-byte, 1-cycle instruction and for a 2-byte, 1-cycle instruction.

Most 8051 instructions execute in one cycle. MUL (multiply) and DIV (divide) are the only instructions that take more than two cycles to complete. They take four cycles.

Normally, two code bytes are fetched from Program Memory during every machine cycle. The only exception to this is when a MOVX instruction is executed. MOVX is a 1-byte 2-cycle instruction that accesses external Data Memory. During a MOVX, two fetches are skipped while the external Data Memory is being addressed and strobed. Figures 7-4C and 7-4D show the timing for a normal 1-byte, 2-cycle instruction and for a MOVX instruction.

## 7.4 PORT STRUCTURES AND OPERATION

All four ports in the 8051 are bidirectional. Each consists of a latch (Special Function Registers P0 through P3), an output driver, and an input buffer.

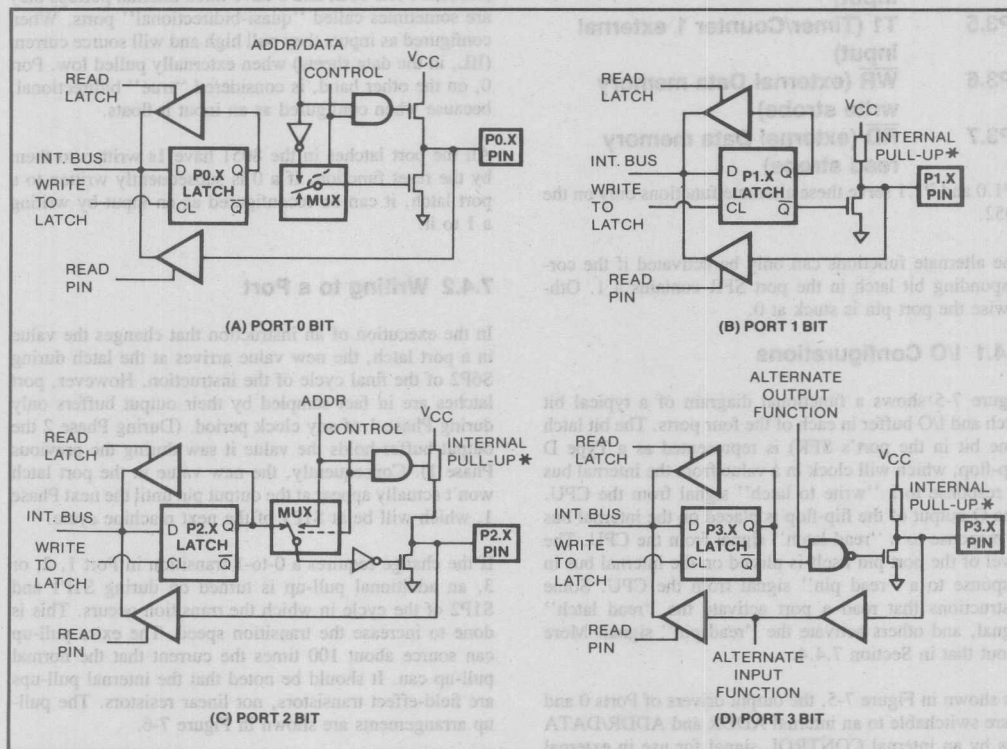


Figure 7-5. 8051 Port Bit Latches and I/O Buffers

\*See Figure 7-6 for details of the internal pullup.

The output drivers of Ports 0 and 2, and the input buffers of Port 0, are used in accesses to external memory. In this application, Port 0 outputs the low byte of the external memory address, time-multiplexed with the byte being written or read. Port 2 outputs the high byte of the external memory address when the address is 16 bits wide. Otherwise the Port 2 pins continue to emit the P2 SFR content.

All the Port 3 pins, and (in the 8052) two Port 1 pins are multifunctional. They are not only port pins, but also serve the functions of various special features as listed below:

| PORT PIN ALTERNATE FUNCTION |   |
|-----------------------------|---|
| *P1.0                       | T2 (Timer/Counter 2 external input)           |
| *P1.1                       | T2EX (Timer/Counter 2 capture/reload trigger) |
| P3.0                        | RXD (serial input port)                       |
| P3.1                        | TXD (serial output port)                      |
| P3.2                        | INT0 (external interrupt)                     |
| P3.3                        | INT1 (external interrupt)                     |
| P3.4                        | T0 (Timer/Counter 0 external input)           |
| P3.5                        | T1 (Timer/Counter 1 external input)           |
| P3.6                        | WR (external Data memory write strobe)        |
| P3.7                        | RD (external Data memory read strobe)         |

\*P1.0 and P1.1 serve these alternate functions only on the 8052.

The alternate functions can only be activated if the corresponding bit latch in the port SFR contains a 1. Otherwise the port pin is stuck at 0.

#### 7.4.1 I/O Configurations

Figure 7-5 shows a functional diagram of a typical bit latch and I/O buffer in each of the four ports. The bit latch (one bit in the port's SFR) is represented as a Type D flip-flop, which will clock in a value from the internal bus in response to a "write to latch" signal from the CPU. The Q output of the flip-flop is placed on the internal bus in response to a "read latch" signal from the CPU. The level of the port pin itself is placed on the internal bus in response to a "read pin" signal from the CPU. Some instructions that read a port activate the "read latch" signal, and others activate the "read pin" signal. More about that in Section 7.4.4.

As shown in Figure 7-5, the output drivers of Ports 0 and 2 are switchable to an internal ADDR and ADDR/DATA bus by an internal CONTROL signal for use in external memory accesses. During external memory accesses, the P2 SFR remains unchanged, but the P0 SFR gets 1s written to it.

Also shown in Figure 7-5, is that if a P3 bit latch contains a 1, then the output level is controlled by the signal labeled "alternate output function." The actual P3.X pin level is always available to the pin's alternate input function, if any.

Ports 1, 2, and 3 have internal pull-ups. Port 0 has open-drain outputs. Each I/O line can be independently used as an input or an output. (Ports 0 and 2 may not be used as general purpose I/O when being used as the ADDR/DATA BUS). To be used as an input, the port bit latch must contain a 1, which turns off the output driver FET. Then, for Ports 1, 2, and 3, the pin is pulled high by the internal pull-up, but can be pulled low by an external source.

Port 0 differs in not having internal pullups. The pullup FET in the P0 output driver (see Figure 7-5A) is used only when the Port is emitting 1s during external memory accesses. Otherwise the pullup FET is off. Consequently P0 lines that are being used as output port lines are open drain. Writing a 1 to the bit latch leaves both output FETs off, so the pin floats. In that condition it can be used as a high-impedance input.

Because Ports 1, 2, and 3 have fixed internal pullups they are sometimes called "quasi-bidirectional" ports. When configured as inputs they pull high and will source current (IIL, in the data sheets) when externally pulled low. Port 0, on the other hand, is considered "true" bidirectional, because when configured as an input it floats.

All the port latches in the 8051 have 1s written to them by the reset function. If a 0 is subsequently written to a port latch, it can be reconfigured as an input by writing a 1 to it.

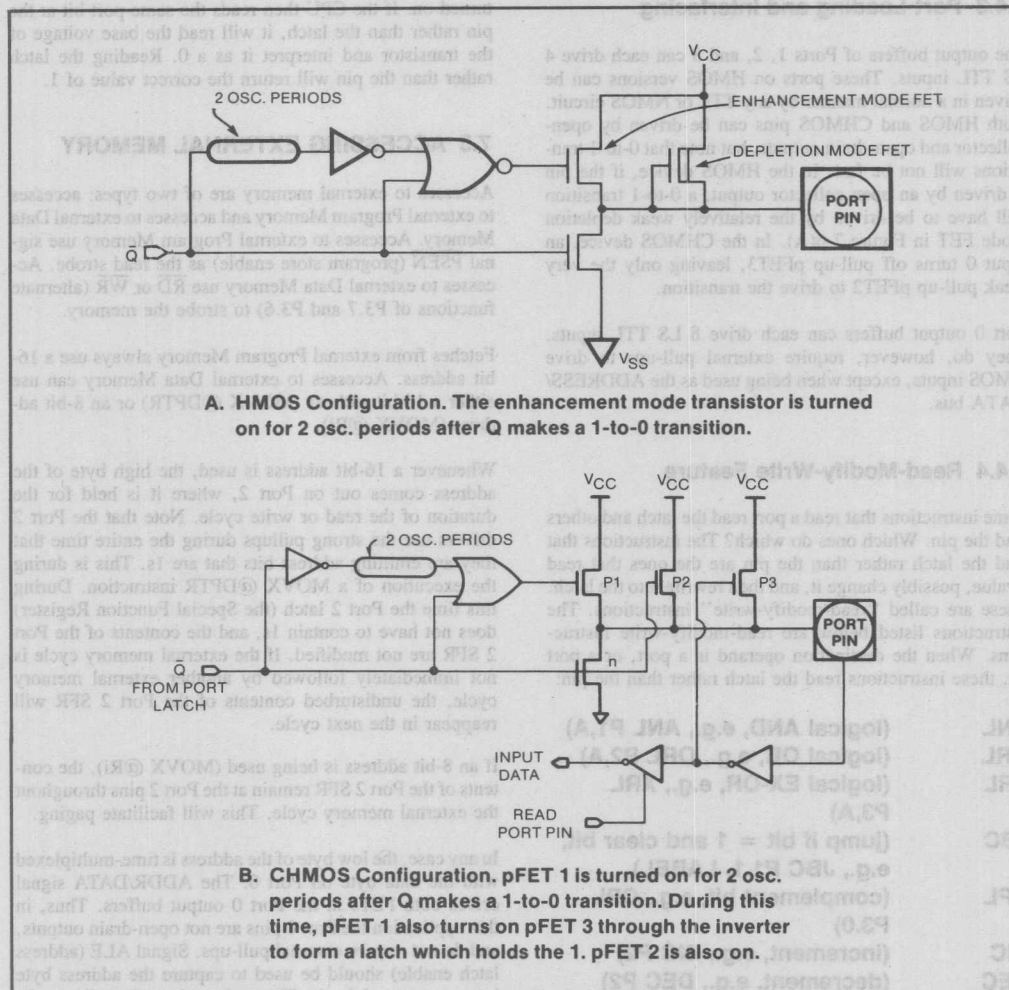
#### 7.4.2 Writing to a Port

In the execution of an instruction that changes the value in a port latch, the new value arrives at the latch during S6P2 of the final cycle of the instruction. However, port latches are in fact sampled by their output buffers only during Phase 1 of any clock period. (During Phase 2 the output buffer holds the value it saw during the previous Phase 1). Consequently, the new value in the port latch won't actually appear at the output pin until the next Phase 1, which will be at S1P1 of the next machine cycle.

If the change requires a 0-to-1 transition in Port 1, 2, or 3, an additional pull-up is turned on during S1P1 and S1P2 of the cycle in which the transition occurs. This is done to increase the transition speed. The extra pull-up can source about 100 times the current that the normal pull-up can. It should be noted that the internal pull-ups are field-effect transistors, not linear resistors. The pull-up arrangements are shown in Figure 7-6.

In HMOS versions of the 8051, the fixed part of the pull-up is a depletion-mode transistor with the gate wired to the source. This transistor will allow the pin to source





**Figure 7-6. Ports 1 and 3 HMOS and CHMOS Internal Pull-up Configurations.**  
**Port 2 is similar except that it holds the strong pullup on while emitting 1s that are address bits.**  
**(See text, "Accessing External Memory.")**

about 0.25 mA when shorted to ground. In parallel with the fixed pull-up is an enhancement-mode transistor, which is activated during S1 whenever the port bit does a 0-to-1 transition. During this interval, if the port pin is shorted to ground, this extra transistor will allow the pin to source an additional 30 mA.

In the CHMOS versions, the pull-up consists of three pFETs. It should be noted that an n-channel FET (nFET) is turned on when a logical 1 is applied to its gate, and is turned off when a logical 0 is applied to its gate. A p-channel FET (pFET) is the opposite: it is on when its gate sees a 0, and off when its gate sees a 1.

pFET 1 in Figure 7-6B is the transistor that is turned on for 2 oscillator periods after a 0-to-1 transition in the port latch. While it's on, it turns on pFET 3 (a weak pull-up), through the inverter. This inverter and pFET form a latch which holds the 1.

Note that if the pin is emitting a 1, a negative glitch on the pin from some external source can turn off pFET 3, causing the pin to go into a float state, pFET 2 is a very weak pull-up which is on whenever the nFET is off, in traditional CMOS style. It's only about 1/10 the strength of pFET3. Its function is to restore a 1 to the pin in the event the pin had a 1 and lost it to a glitch.

### 7.4.3 Port Loading and Interfacing

The output buffers of Ports 1, 2, and 3 can each drive 4 LS TTL inputs. These ports on HMOS versions can be driven in a normal manner by any TTL or NMOS circuit. Both HMOS and CHMOS pins can be driven by open-collector and open-drain outputs, but note that 0-to-1 transitions will not be fast. In the HMOS device, if the pin is driven by an open collector output, a 0-to-1 transition will have to be driven by the relatively weak depletion mode FET in Figure 7-6(A). In the CHMOS device, an input 0 turns off pull-up pFET3, leaving only the very weak pull-up pFET2 to drive the transition.

Port 0 output buffers can each drive 8 LS TTL inputs. They do, however, require external pull-ups to drive NMOS inputs, except when being used as the ADDRESS/DATA bus.

### 7.4.4 Read-Modify-Write Feature

Some instructions that read a port read the latch and others read the pin. Which ones do which? The instructions that read the latch rather than the pin are the ones that read a value, possibly change it, and then rewrite it to the latch. These are called "read-modify-write" instructions. The instructions listed below are read-modify-write instructions. When the destination operand is a port, or a port bit, these instructions read the latch rather than the pin:

|                   |  |
|-------------------|--|
| <b>ANL</b>        | (logical AND, e.g., ANL P1,A)                          |
| <b>ORL</b>        | (logical OR, e.g., ORL P2,A)                           |
| <b>XRL</b>        | (logical EX-OR, e.g., XRL P3,A)                        |
| <b>JBC</b>        | (jump if bit = 1 and clear bit, e.g., JBC P1.1, LABEL) |
| <b>CPL</b>        | (complement bit, e.g., CPL P3.0)                       |
| <b>INC</b>        | (increment, e.g., INC P2)                              |
| <b>DEC</b>        | (decrement, e.g., DEC P2)                              |
| <b>DJNZ</b>       | (decrement and jump if not zero, e.g., DJNZ P3, LABEL) |
| <b>MOV PX,Y,C</b> | (move carry bit to bit Y of Port X)                    |
| <b>CLR PX.Y</b>   | (clear bit Y of Port X)                                |
| <b>SET PX.Y</b>   | (set bit Y of Port X)                                  |

It is not obvious that the last three instructions in this list are read-modify-write instructions, but they are. They read the port byte, all 8 bits, modify the addressed bit, then write the new byte back to the latch.

The reason that read-modify-write instructions are directed to the latch rather than the pin is to avoid a possible misinterpretation of the voltage level at the pin. For example, a port bit might be used to drive the base of a transistor. When a 1 is written to the bit, the transistor is

turned on. If the CPU then reads the same port bit at the pin rather than the latch, it will read the base voltage of the transistor and interpret it as a 0. Reading the latch rather than the pin will return the correct value of 1.

### 7.5 ACCESSING EXTERNAL MEMORY

Accesses to external memory are of two types: accesses to external Program Memory and accesses to external Data Memory. Accesses to external Program Memory use signal  $\overline{\text{PSEN}}$  (program store enable) as the read strobe. Accesses to external Data Memory use  $\overline{\text{RD}}$  or  $\overline{\text{WR}}$  (alternate functions of P3.7 and P3.6) to strobe the memory.

Fetches from external Program Memory always use a 16-bit address. Accesses to external Data Memory can use either a 16-bit address ( $\text{MOVX @DPTR}$ ) or an 8-bit address ( $\text{MOVX @Ri}$ ).

Whenever a 16-bit address is used, the high byte of the address comes out on Port 2, where it is held for the duration of the read or write cycle. Note that the Port 2 drivers use the strong pullups during the entire time that they are emitting address bits that are 1s. This is during the execution of a  $\text{MOVX @DPTR}$  instruction. During this time the Port 2 latch (the Special Function Register) does not have to contain 1s, and the contents of the Port 2 SFR are not modified. If the external memory cycle is not immediately followed by another external memory cycle, the undisturbed contents of the Port 2 SFR will reappear in the next cycle.

If an 8-bit address is being used ( $\text{MOVX @Ri}$ ), the contents of the Port 2 SFR remain at the Port 2 pins throughout the external memory cycle. This will facilitate paging.

In any case, the low byte of the address is time-multiplexed with the data byte on Port 0. The ADDR/DATA signal drives both FETs in the Port 0 output buffers. Thus, in this application the Port 0 pins are not open-drain outputs, and do not require external pull-ups. Signal ALE (address latch enable) should be used to capture the address byte into an external latch. The address byte is valid at the negative transition of ALE. Then, in a write cycle, the data byte to be written appears on Port 0 just before  $\overline{\text{WR}}$  is activated, and remains there until after  $\overline{\text{WR}}$  is deactivated. In a read cycle, the incoming byte is accepted at Port 0 just before the read strobe is deactivated.

During any access to external memory, the CPU writes 0FFH to the Port 0 latch (the Special Function Register), thus obliterating whatever information the Port 0 SFR may have been holding.

External Program Memory is accessed under two conditions:

- 1) Whenever signal  $\overline{\text{EA}}$  is active; or
- 2) Whenever the program counter (PC) contains a number that is larger than 0FFFH (1FFFH for the 8052).

This requires that the ROMless versions have  $\overline{\text{EA}}$  wired

low to enable the lower 4K (8K for the 8032) program bytes to be fetched from external memory.

When the CPU is executing out of external Program Memory, all 8 bits of Port 2 are dedicated to an output function and may not be used for general purpose I/O. During external program fetches they output the high byte of the PC. During this time the Port 2 drivers use the strong pullups to emit PC bits that are 1s.

### 7.5.1 PSEN

The read strobe for external fetches is  $\overline{\text{PSEN}}$ .  $\overline{\text{PSEN}}$  is not activated for internal fetches. When the CPU is accessing external Program Memory,  $\overline{\text{PSEN}}$  is activated twice every cycle (except during a MOVX instruction) whether or not the byte fetched is actually needed for the current instruction. When  $\overline{\text{PSEN}}$  is activated its timing is not the same as RD. A complete RD cycle, including activation and deactivation of ALE and RD, takes 12

oscillator periods. A complete  $\overline{\text{PSEN}}$  cycle, including activation and deactivation of ALE and  $\overline{\text{PSEN}}$ , takes 6 oscillator periods. The execution sequence for these two types of read cycles are shown in Figure 7-7 for comparison.

### 7.5.2 ALE

The main function of ALE is to provide a properly timed signal to latch the low byte of an address from P0 to an external latch during fetches from external Program Memory. For that purpose ALE is activated twice every machine cycle. This activation takes place even when the cycle involves no external fetch. The only time an ALE pulse doesn't come out is during an access to external Data Memory. The first ALE of the second cycle of a MOVX instruction is missing (see Figure 7-7). Consequently, in any system that does not use external Data Memory, ALE is activated at a constant rate of 1/6 the oscillator frequency, and can be used for external clocking or timing purposes.

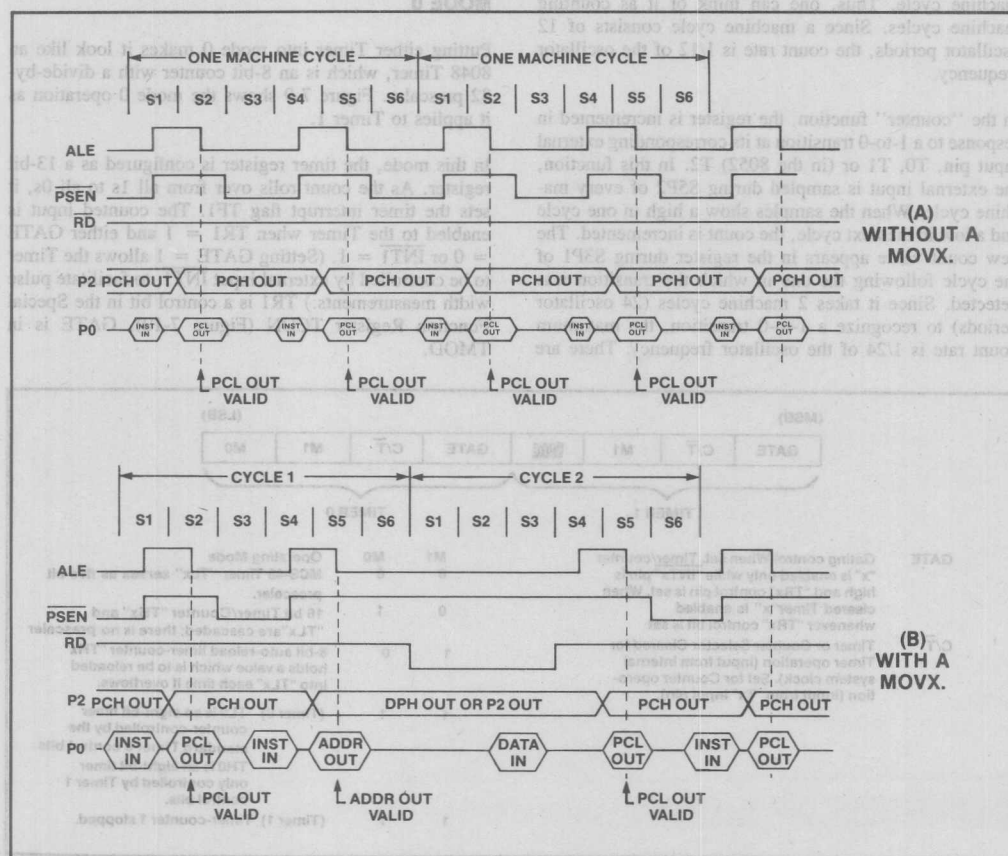


Figure 7-7. External Program Memory Execution

### 7.5.3 Overlapping External Program and Data Memory Spaces

In some applications it is desirable to execute a program from the same physical memory that is being used to store data. In the 8051, the external Program and Data Memory spaces can be combined by ANDing PSEN and RD. A positive-logic AND of these two signals produces an active-low read strobe that can be used for the combined physical memory. Since the PSEN cycle is faster than the RD cycle, the external memory needs to be fast enough to accommodate the PSEN cycle.

## 7.6 TIMER/COUNTERS

The 8051 has two 16-bit timer/counter registers: Timer 0 and Timer 1. The 8052 has these two plus one more: Timer 2. All three can be configured to operate either as timers or event counters.

In the "timer" function, the register is incremented every machine cycle. Thus, one can think of it as counting machine cycles. Since a machine cycle consists of 12 oscillator periods, the count rate is 1/12 of the oscillator frequency.

In the "counter" function, the register is incremented in response to a 1-to-0 transition at its corresponding external input pin, T0, T1 or (in the 8052) T2. In this function, the external input is sampled during SSP2 of every machine cycle. When the samples show a high in one cycle and a low in the next cycle, the count is incremented. The new count value appears in the register during S3P1 of the cycle following the one in which the transition was detected. Since it takes 2 machine cycles (24 oscillator periods) to recognize a 1-to-0 transition, the maximum count rate is 1/24 of the oscillator frequency. There are

no restrictions on the duty cycle of the external input signal, but to ensure that a given level is sampled at least once before it changes, it should be held for at least one full machine cycle.

In addition to the "timer" or "counter" selection, Timer 0 and Timer 1 have four operating modes from which to select. Timer 2, in the 8052, has three modes of operation: "capture," "auto-reload" and "baud rate generator."

### 7.6.1 Timer 0 and Timer 1

These timer/counters are present in both the 8051 and the 8052. The "timer" or "counter" function is selected by control bits C/T in the Special Function Register TMOD (Figure 6-8). These two timer/counters have four operating modes, which are selected by bit-pairs (M1, M0) in TMOD. Modes 0, 1, and 2 are the same for both timer/counters. Mode 3 is different. The four operating modes are described below.

#### MODE 0

Putting either Timer into mode 0 makes it look like an 8048 Timer, which is an 8-bit counter with a divide-by-32 prescaler. Figure 7-9 shows the mode 0 operation as it applies to Timer 1.

In this mode, the timer register is configured as a 13-bit register. As the count rolls over from all 1s to all 0s, it sets the timer interrupt flag TF1. The counted input is enabled to the Timer when TR1 = 1 and either GATE = 0 or INT1 = 1. (Setting GATE = 1 allows the Timer to be controlled by external input INT1, to facilitate pulse width measurements.) TR1 is a control bit in the Special Function Register TCON (Figure 7-10). GATE is in TMOD.

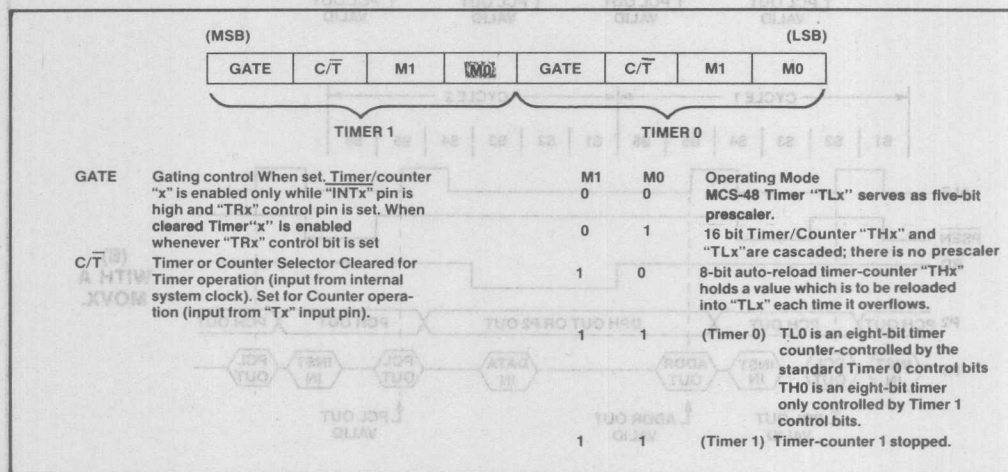


Figure 7-8. TMOD: Timer/Counter Mode Control Register



The 13-bit register consists of all 8 bits of TH1 and the lower 5 bits of TL1. The upper 3 bits of TL1 are indeterminate and should be ignored. Setting the run flag (TR1) does not clear the registers.

Mode 0 operation is the same for Timer 0 as for Timer 1. Substitute TR0, TF0 and INT0 for the corresponding Timer 1 signals in Figure 7-9. There are two different GATE bits, one for Timer 1 (TMOD.7) and one for Timer 0 (TMOD.3).

#### MODE 1

Mode 1 is the same as Mode 0, except that the Timer register is being run with all 16 bits.

#### MODE 2

Mode 2 configures the timer register as an 8-bit counter (TL1) with automatic reload, as shown in Figure 7-11. Overflow from TL1 not only sets TF1, but also reloads TL1 with the contents of TH1, which is preset by software. The reload leaves TH1 unchanged.

Mode 2 operation is the same for Timer/Counter 0.

#### MODE 3

Timer 1 in Mode 3 simply holds its count. The effect is the same as setting TR1 = 0.

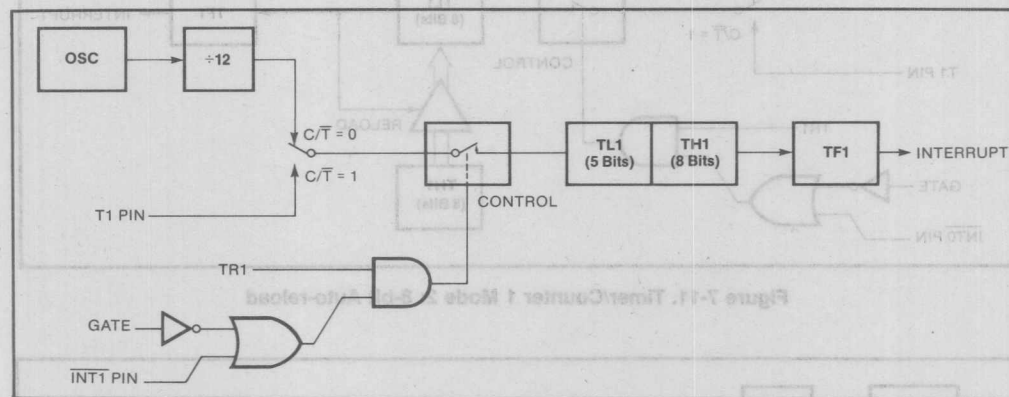


Figure 7-9. Timer/Counter 1 Mode 0: 13-bit Counter

| (MSB)  |          |  |     | (LSB)  |          |  |     |
|--------|----------|--|-----|--------|----------|--|-----|
| TF1    | TR1      | TF0  | TR0 | IE1    | IT1      | IE0  | IT0 |
| Symbol | Position | Name and Significance  |     | Symbol | Position | Name and Significance  |     |
| TF1    | TCON.7   | Timer 1 overflow flag. Set by hardware on timer/counter overflow. Cleared by hardware when processor vectors to interrupt routine. |     | IE1    | TCON.3   | Interrupt 1 Edge flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.        |     |
| TR1    | TCON.6   | Timer 1 Run control bit. Set/cleared by software to turn timer/counter on/off.   |     | IT1    | TCON.2   | Interrupt 1 Type control bit. Set/cleared by software to specify falling edge/low level triggered external interrupts. |     |
| TF0    | TCON.5   | Timer 0 overflow flag. Set by hardware on timer/counter overflow. Cleared by hardware when processor vectors to interrupt routine. |     | IE0    | TCON.1   | Interrupt 0 Edge flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.        |     |
| TR0    | TCON.4   | Timer 0 Run control bit. Set/cleared by software to turn timer/counter on/off.   |     | IT0    | TCON.0   | Interrupt 0 Type control bit. Set/cleared by software to specify falling edge/low level triggered external interrupts. |     |

Figure 7-10. TCON: Timer/Counter Control Register

Timer 0 in Mode 3 establishes TL0 and TH0 as two separate counters. The logic for Mode 3 on Timer 0 is shown in Figure 7-12. TL0 uses the Timer 0 control bits:  $C/\bar{T}$ , GATE, TR0, INT0, and TF0. TH0 is locked into a timer function (counting machine cycles) and takes over the use of TR1 and TF1 from Timer 1. Thus, TH0 now controls the "Timer 1" interrupt.

Mode 3 is provided for applications requiring an extra 8-bit timer or counter. With Timer 0 in Mode 3, an 8051 can look like it has three timer/counters, and an 8052, like it has four. When Timer 0 is in Mode 3, Timer 1 can be

turned on and off by switching it out of and into its own Mode 3, or can still be used by the serial port as a baud rate generator, or in fact, in any application not requiring an interrupt.

### 7.6.2 Timer 2

Timer 2 is a 16-bit timer/counter which is present only in the 8052. Like Timers 0 and 1, it can operate either as a timer or as an event counter. This is selected by bit  $C/T2$  in the Special Function Register T2CON (Figure 7-13). It has three operating modes: "capture," "auto-

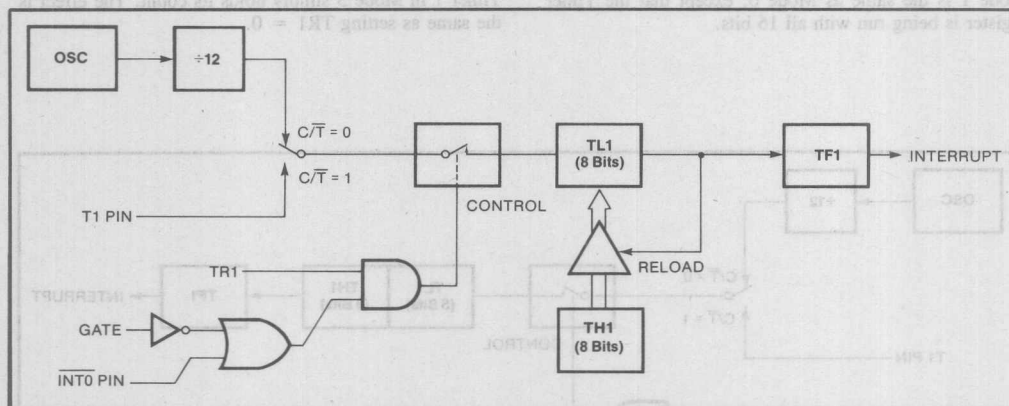


Figure 7-11. Timer/Counter 1 Mode 2: 8-bit Auto-reload

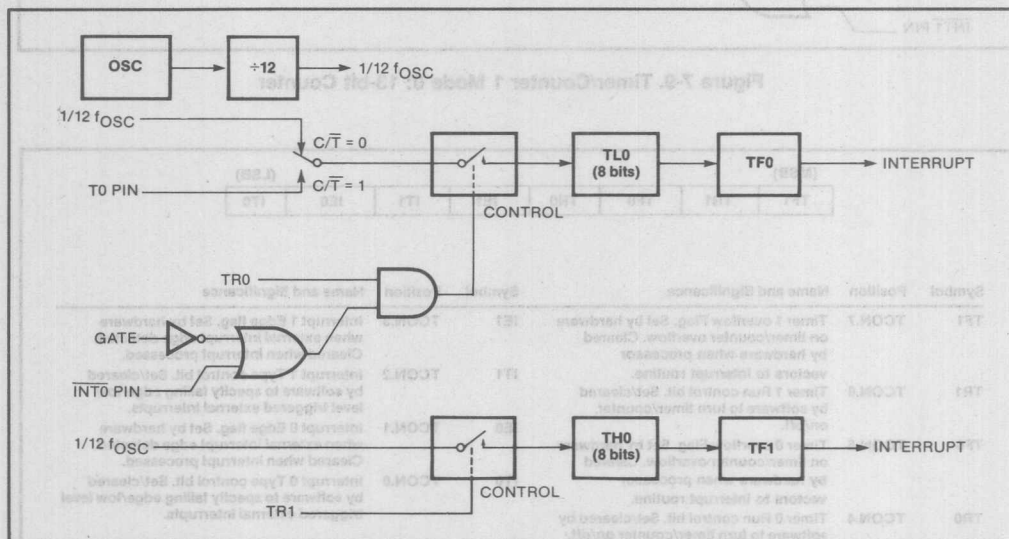


Figure 7-12. Timer/Counter 0 Mode 3: Two 8-bit Counters

| (MSB)  |          |  |      | (LSB) |     |      |        |
|--------|----------|--|------|-------|-----|------|--------|
| TF2    | EXF2     | RCLK   | TCLK | EXEN2 | TR2 | C/T2 | CP/RL2 |
| Symbol | Position | Name and Significance  |      |       |     |      |        |
| TF2    | T2CON.7  | Timer 2 overflow flag set by a Timer 2 overflow and must be cleared by software. TF2 will not be set when either RCLK = 1 or TCLK = 1.   |      |       |     |      |        |
| EXF2   | T2CON.6  | Timer 2 external flag set when either a capture or reload is caused by a negative transition on T2EX and EXEN2 = 1. When Timer 2 interrupt is enabled, EXF2 = 1 will cause the CPU to vector to the Timer 2 interrupt routine. EXF2 must be cleared by software.   |      |       |     |      |        |
| RCLK   | T2CON.5  | Receive clock flag. When set, causes the serial port to use Timer 2 overflow pulses for its receive clock in modes 1 and 3. RCLK = 0 causes Timer 1 overflow to be used for the receive clock.   |      |       |     |      |        |
| TCLK   | T2CON.4  | Transmit clock flag. When set, causes the serial port to use Timer 2 overflow pulses for its transmit clock in modes 1 and 3. TCLK = 0 causes Timer 1 overflows to be used for the transmit clock.   |      |       |     |      |        |
| EXEN2  | T2CON.3  | Timer 2 external enable flag. When set, allows a capture or reload to occur as a result of a negative transition on T2EX if Timer 2 is not being used to clock the serial port. EXEN2 = 0 causes Timer 2 to ignore events at T2EX.   |      |       |     |      |        |
| TR2    | T2CON.2  | Start/stop control for Timer 2. A logic 1 starts the timer.  |      |       |     |      |        |
| C/T2   | T2CON.1  | Timer or counter select. (Timer 2)<br>0 = Internal timer (OSC/12)<br>1 = External event counter (falling edge triggered).  |      |       |     |      |        |
| CP/RL2 | T2CON.0  | Capture/Reload flag. When set, captures will occur on negative transitions at T2EX if EXEN2 = 1. When cleared, auto reloads will occur either with Timer 2 overflows or negative transitions at T2EX when EXEN2 = 1. When either RCLK = 1 or TCLK = 1, this bit is ignored and the timer is forced to auto-reload on Timer 2 overflow. |      |       |     |      |        |

Figure 7-13. T2CON: Timer/Counter 2 Control Register

load" and "baud rate generator," which are selected by bits in T2CON as shown in Table 2.

Table 2. Timer 2 Operating Modes

| RCLK + TCLK | CP/RL2 | TR2 | MODE                |
|-------------|--------|-----|---------------------|
| 0           | 0      | 1   | 16-bit auto-reload  |
| 0           | 1      | 1   | 16-bit capture      |
| 1           | X      | 1   | baud rate generator |
| X           | X      | 0   | (off)               |

In the capture mode there are two options which are selected by bit EXEN2 in T2CON. If EXEN2 = 0, then Timer 2 is a 16-bit timer or counter which upon overflowing sets bit TF2, the Timer 2 overflow bit, which can be used to generate an interrupt. If EXEN2 = 1, then Timer 2 still does the above, but with the added feature that a 1-to-0 transition at external input T2EX causes the current value in the Timer 2 registers, TL2 and TH2, to be captured into registers RCAP2L and RCAP2H, respectively. (RCAP2L and RCAP2H are new Special Function Registers in the 8052.) In addition, the transition at T2EX causes bit EXF2 in T2CON to be set, and EXF2, like TF2, can generate an interrupt.

The capture mode is illustrated in Figure 7-14.

In the auto-reload mode there are again two options, which are selected by bit EXEN2 in T2CON. If EXEN2 = 0, then when Timer 2 rolls over it not only sets TF2 but also causes the Timer 2 registers to be reloaded with the 16-bit value in registers RCAP2L and RCAP2H, which are preset by software. If EXEN2 = 1, then Timer 2 still does the above, but with the added feature that a 1-to-0 transition at external input T2EX will also trigger the 16-bit reload and set EXF2.

The auto-reload mode is illustrated in Figure 7-15.

The baud rate generator mode is selected by RCLK = 1 and/or TCLK = 1. It will be described in conjunction with the serial port.

## 7.7 SERIAL INTERFACE

The serial port is full duplex, meaning it can transmit and receive simultaneously. It is also receive-buffered, meaning it can commence reception of a second byte before a previously received byte has been read from the receive register. (However, if the first byte still hasn't been read by the time reception of the second byte is complete, one of the bytes will be lost). The serial port receive and transmit registers are both accessed at Special Function Register SBUF. Writing to SBUF loads the transmit reg-

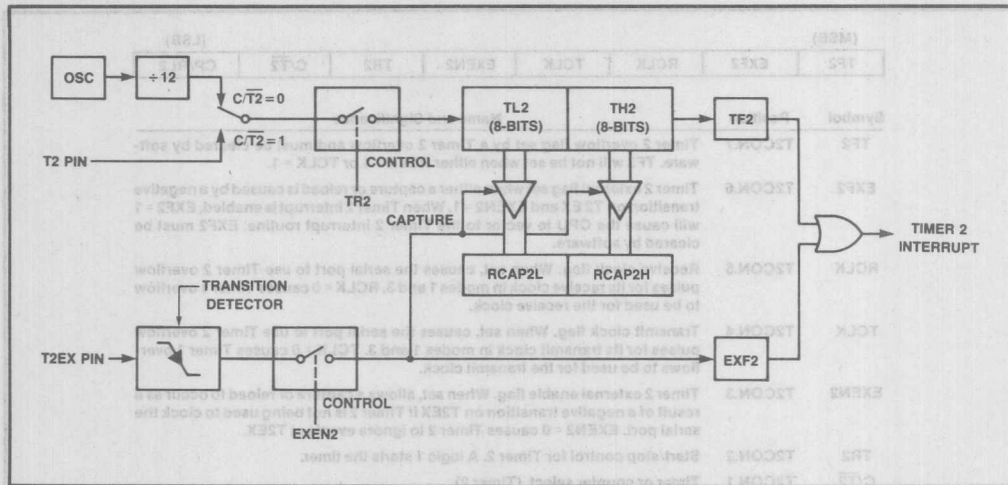


Figure 7-14. Timer 2 in Capture Mode

ister, and reading SBUF accesses a physically separate receive register.

The serial port can operate in 4 modes:

**Mode 0:** Serial data enters and exits through RXD. TXD outputs the shift clock. 8 bits are transmitted/received: 8 data bits (LSB first). The baud rate is fixed at 1/12 the oscillator frequency.

**Mode 1:** 10 bits are transmitted (through TXD) or received (through RXD): a start bit (0), 8 data bits (LSB first), and a stop bit (1). On receive, the stop bit goes into RB8 in Special Function Register SCON. The baud rate is variable.

**Mode 2:** 11 bits are transmitted (through TXD) or received (through RXD): a start bit (0), 8 data bits (LSB first), a programmable 9th data bit, and a stop bit (1). On transmit, the 9th data bit (TB8 in SCON) can be assigned the value of 0 or 1. Or, for example, the parity bit (P, in the PSW) could be moved into TB8. On receive, the 9th data bit goes into RB8 in Special Function Register SCON, while the stop bit is ignored. The baud rate is programmable to either 1/32 or 1/64 the oscillator frequency.

**Mode 3:** 11 bits are transmitted (through TXD) or received (through RXD): a start bit (0), 8 data bits (LSB first), a programmable 9th data bit and a stop bit (1). In fact, Mode 3 is the same as Mode 2 in all respects except the baud rate. The baud rate in Mode 3 is variable.

In all four modes, transmission is initiated by any instruction that uses SBUF as a destination register. Reception is initiated in Mode 0 by the condition RI = 0 and REN = 1. Reception is initiated in the other modes by the incoming start bit if REN = 1.

## 7.7.1 Multiprocessor Communications

Modes 2 and 3 have a special provision for multiprocessor communications. In these modes, 9 data bits are received. The 9th one goes into RB8. Then comes a stop bit. The port can be programmed such that when the stop bit is received, the serial port interrupt will be activated only if RB8 = 1. This feature is enabled by setting bit SM2 in SCON. A way to use this feature in multiprocessor systems is as follows.

When the master processor wants to transmit a block of data to one of several slaves, it first sends out an address byte which identifies the target slave. An address byte differs from a data byte in that the 9th bit is 1 in an address byte and 0 in a data byte. With SM2 = 1, no slave will be interrupted by a data byte. An address byte, however, will interrupt all slaves, so that each slave can examine the received byte and see if it is being addressed. The addressed slave will clear its SM2 bit and prepare to receive the data bytes that will be coming. The slaves that weren't being addressed leave their SM2s set and go on about their business, ignoring the coming data bytes.

SM2 has no effect in Mode 0, and in Mode 1 can be used to check the validity of the stop bit. In a Mode 1 reception, if SM2 = 1, the receive interrupt will not be activated unless a valid stop bit is received.

## 7.7.2 Serial Port Control Register

The serial port control and status register is the Special Function Register SCON, shown in Figure 7-16. This register contains not only the mode selection bits, but also the 9th data bit for transmit and receive (TB8 and RB8), and the serial port interrupt bits (TI and RI).



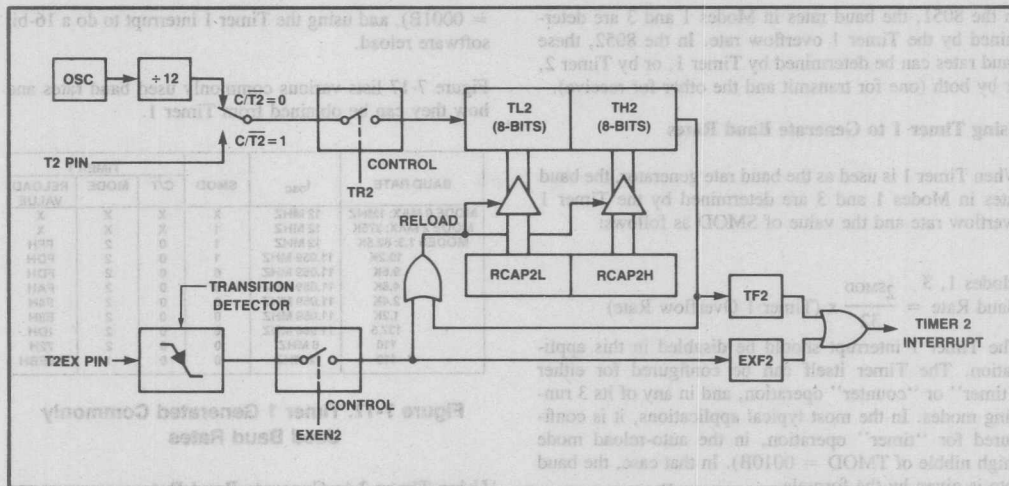


Figure 7-15. Timer 2 in Auto-Reload Mode

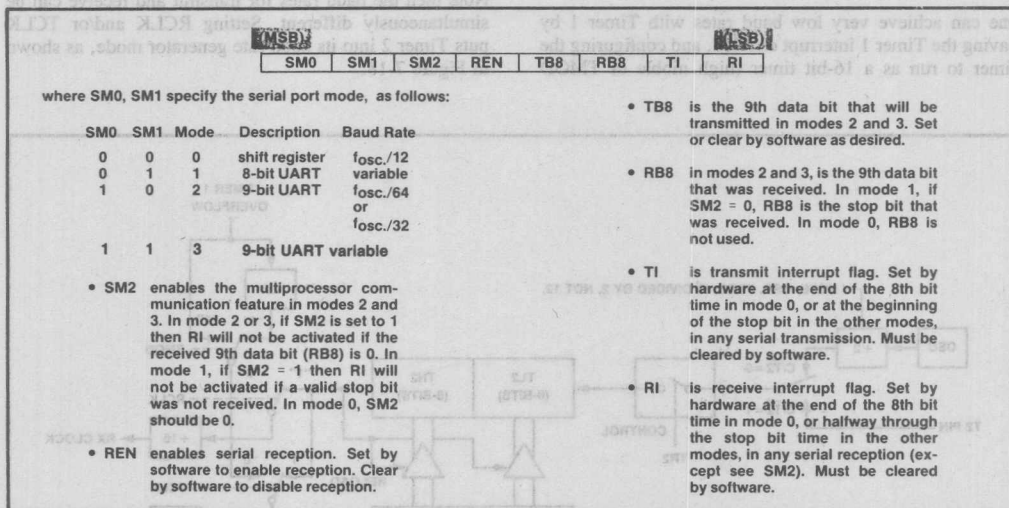


Figure 7-16. SCON: Serial Port Control Register

### 7.7.3 Baud Rates

The baud rate in Mode 0 is fixed:

$$\text{Mode 0 Baud Rate} = \frac{\text{Oscillator Frequency}}{12}$$

The baud rate in Mode 2 depends on the value of bit SMOD in Special Function Register PCON. If SMOD = 0 (which is its value on reset), the baud rate is 1/64 the oscillator frequency. If SMOD = 1, the baud rate is 1/32 the oscillator frequency.

$$\text{Mode 2 Baud Rate} = \frac{2^{\text{SMOD}}}{64} \times (\text{Oscillator Frequency})$$

In the 8051, the baud rates in Modes 1 and 3 are determined by the Timer 1 overflow rate. In the 8052, these baud rates can be determined by Timer 1, or by Timer 2, or by both (one for transmit and the other for receive).

#### Using Timer 1 to Generate Baud Rates

When Timer 1 is used as the baud rate generator, the baud rates in Modes 1 and 3 are determined by the Timer 1 overflow rate and the value of SMOD as follows:

$$\text{Baud Rate} = \frac{2^{\text{SMOD}}}{32} \times (\text{Timer 1 Overflow Rate})$$

The Timer 1 interrupt should be disabled in this application. The Timer itself can be configured for either "timer" or "counter" operation, and in any of its 3 running modes. In the most typical applications, it is configured for "timer" operation, in the auto-reload mode (high nibble of TMOD = 0010B). In that case, the baud rate is given by the formula

$$\text{Modes 1, 3 Baud Rate} = \frac{2^{\text{SMOD}}}{32} \times \frac{\text{Oscillator Frequency}}{12 \times [256 - (\text{TH1})]}$$

One can achieve very low baud rates with Timer 1 by leaving the Timer 1 interrupt enabled, and configuring the Timer to run as a 16-bit timer (high nibble of TMOD

= 0001B), and using the Timer 1 interrupt to do a 16-bit software reload.

Figure 7-17 lists various commonly used baud rates and how they can be obtained from Timer 1.

| BAUD RATE        | f <sub>osc</sub> | SMOD | TIMER 1 |      |              |
|------------------|------------------|------|---------|------|--------------|
|                  |                  |      | C/T     | MODE | RELOAD VALUE |
| MODE 0 MAX: 1MHZ | 12 MHZ           | X    | X       | X    | X            |
| MODE 2 MAX: 375K | 12 MHZ           | 1    | X       | X    | X            |
| MODES 1,3: 62.5K | 12 MHZ           | 1    | 0       | 2    | FFH          |
| 19.2K            | 11.059 MHZ       | 1    | 0       | 2    | FDH          |
| 9.6K             | 11.059 MHZ       | 0    | 0       | 2    | FAH          |
| 4.8K             | 11.059 MHZ       | 0    | 0       | 2    | F4H          |
| 2.4K             | 11.059 MHZ       | 0    | 0       | 2    | E8H          |
| 1.2K             | 11.059 MHZ       | 0    | 0       | 2    | 1DH          |
| 137.5            | 11.986 MHZ       | 0    | 0       | 2    | 72H          |
| 110              | 6 MHZ            | 0    | 0       | 1    | FE6BH        |
| 110              | 12 MHZ           | 0    | 0       | 1    |              |

Figure 7-17. Timer 1 Generated Commonly Used Baud Rates

#### Using Timer 2 to Generate Baud Rates

In the 8052, Timer 2 is selected as the baud rate generator by setting TCLK and/or RCLK in T2CON (Figure 7-13). Note then the baud rates for transmit and receive can be simultaneously different. Setting RCLK and/or TCLK puts Timer 2 into its baud rate generator mode, as shown in Figure 7-18.

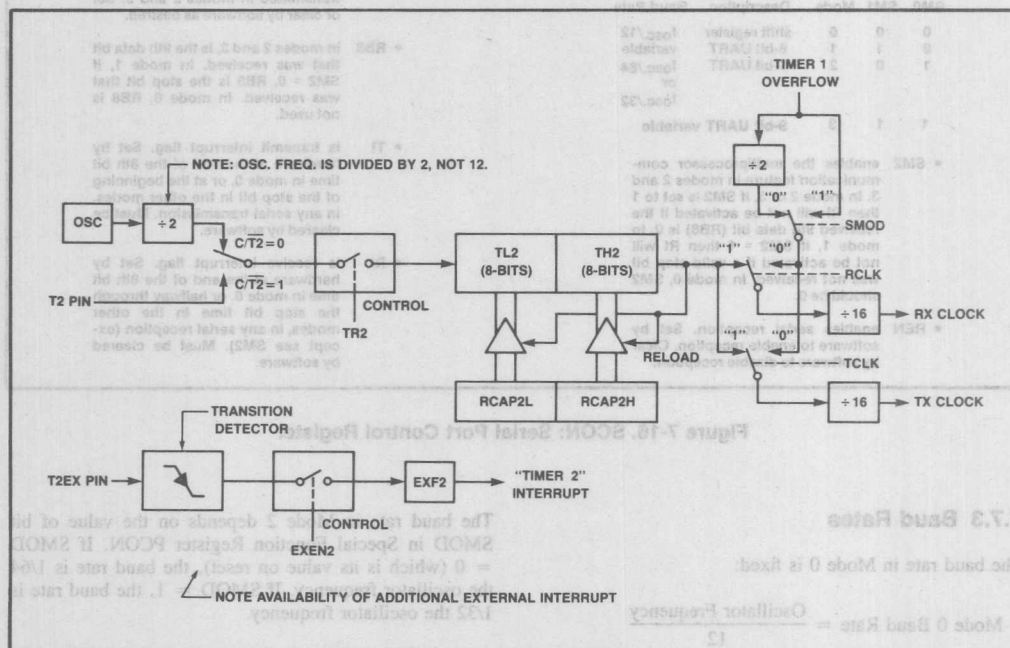


Figure 7-18. Timer 2 in Baud Rate Generator Mode

The baud rate generator mode is similar to the auto-reload mode, in that a rollover in TH2 causes the Timer 2 registers to be reloaded with the 16-bit value in registers RCAP2H and RCAP2L, which are preset by software.

Now, the baud rates in Modes 1 and 3 are determined by Timer 2's overflow rate as follows:

$$\text{Modes 1, 3 Baud Rate} = \frac{\text{Timer 2 Overflow Rate}}{16}$$

The Timer can be configured for either "timer" or "counter" operation. In the most typical applications, it is configured for "timer" operation ( $C/\overline{T2} = 0$ ). "Timer" operation is a little different for Timer 2 when it's being used as a baud rate generator. Normally as a timer it would increment every machine cycle (thus at 1/12 the oscillator frequency). As a baud rate generator, however, it increments every state time (thus at 1/2 the oscillator frequency). In that case the baud rate is given by the formula

$$\text{Modes 1, 3 Baud Rate} = \frac{\text{Oscillator Frequency}}{32 \times [65536 - (\text{RCAP2H}, \text{RCAP2L})]}$$

where (RCAP2H, RCAP2L) is the content of RCAP2H and RCAP2L taken as a 16-bit unsigned integer.

Timer 2 as a baud rate generator is shown in Figure 7-18. This Figure is valid only if  $\text{RCLK} + \text{TCLK} = 1$  in T2CON. Note that a rollover in TH2 does not set TF2, and will not generate an interrupt. Therefore, the Timer 2 interrupt does not have to be disabled when Timer 2 is in the baud rate generator mode. Note too, that if EXEN2 is set, a 1-to-0 transition in T2EX will set EXF2 but will not cause a reload from (RCAP2H, RCAP2L) to (TH2, TL2). Thus when Timer 2 is in use as a baud rate generator, T2EX can be used as an extra external interrupt, if desired.

It should be noted that when Timer 2 is running ( $\text{TR2} = 1$ ) in "timer" function in the baud rate generator mode, one should not try to read or write TH2 or TL2. Under these conditions the Timer is being incremented every state time, and the results of a read or write may not be accurate. The RCAP registers may be read, but shouldn't be written to, because a write might overlap a reload and cause write and/or reload errors. Turn the Timer off (clear TR2) before accessing the Timer 2 or RCAP registers, in this case.

#### 7.7.4 More About Mode 0

Serial data enters and exits through RXD. TXD outputs the shift clock. 8 bits are transmitted/received: 8 data bits (LSB first). The baud rate is fixed at 1/12 the oscillator frequency.

Figure 7-19 shows a simplified functional diagram of the serial port in mode 0, and associated timing.

Transmission is initiated by any instruction that uses SBUF as a destination register. The "write to SBUF" signal at S6P2 also loads a 1 into the 9th bit position of the transmit shift register and tells the TX Control block to commence a transmission. The internal timing is such that one full machine cycle will elapse between "write to SBUF," and activation of SEND.

SEND enables the output of the shift register to the alternate output function line of P3.0, and also enables SHIFT CLOCK to the alternate output function line of P3.1. SHIFT CLOCK is low during S3, S4, and S5 of every machine cycle, and high during S6, S1 and S2. At S6P2 of every machine cycle in which SEND is active, the contents of the transmit shift register are shifted to the right one position.

As data bits shift out to the right, zeros come in from the left. When the MSB of the data byte is at the output position of the shift register, then the 1 that was initially loaded into the 9th position, is just to the left of the MSB, and all positions to the left of that contain zeros. This condition flags the TX Control block to do one last shift and then deactivate SEND and set T1. Both of these actions occur at S1P1 of the 10th machine cycle after "write to SBUF."

Reception is initiated by the condition  $\text{REN} = 1$  and  $\text{RI} = 0$ . At S6P2 of the next machine cycle, the RX Control unit writes the bits 11111110 to the receive shift register, and in the next clock phase activates RECEIVE.

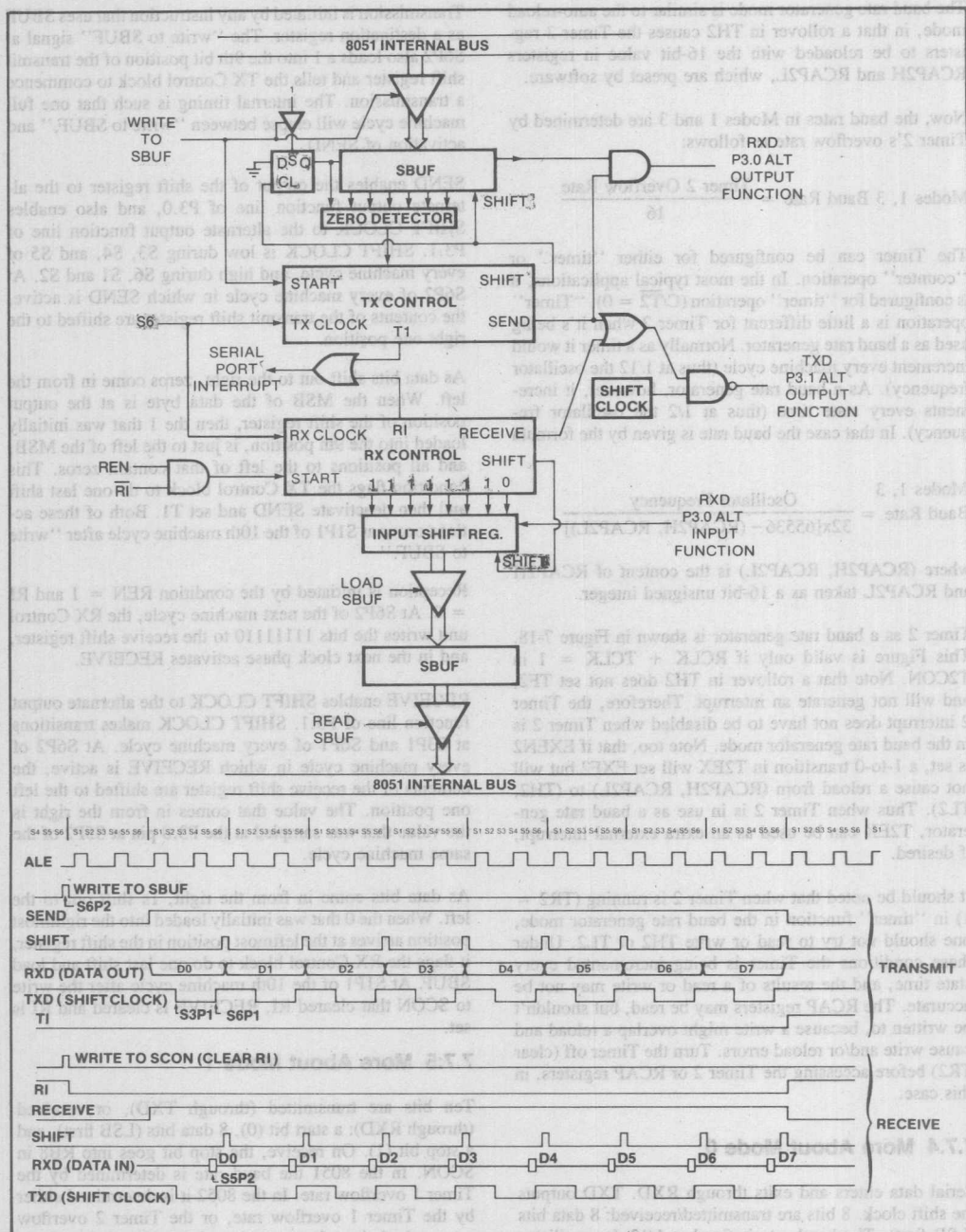
RECEIVE enables SHIFT CLOCK to the alternate output function line of P3.1. SHIFT CLOCK makes transitions at S3P1 and S6P1 of every machine cycle. At S6P2 of every machine cycle in which RECEIVE is active, the contents of the receive shift register are shifted to the left one position. The value that comes in from the right is the value that was sampled at the P3.0 pin at S5P2 of the same machine cycle.

As data bits come in from the right, 1s shift out to the left. When the 0 that was initially loaded into the rightmost position arrives at the leftmost position in the shift register, it flags the RX Control block to do one last shift and load SBUF. At S1P1 of the 10th machine cycle after the write to SCON that cleared RI, RECEIVE is cleared and RI is set.

#### 7.7.5 More About Mode 1

Ten bits are transmitted (through TXD), or received (through RXD): a start bit (0), 8 data bits (LSB first), and a stop bit (1). On receive, the stop bit goes into RB8 in SCON. In the 8051 the baud rate is determined by the Timer 1 overflow rate. In the 8052 it is determined either by the Timer 1 overflow rate, or the Timer 2 overflow rate, or both (one for transmit and the other for receive).

Figure 7-20 shows a simplified functional diagram of the serial port in Mode 1, and associated timings for transmit and receive.



**Figure 7-19. Serial Port Mode 0**



[illegible]

**Figure 7-20. Serial Port Mode 1**

**TCLK, RCLK, and Timer 2 are present in the 8052/8032 only.**

Transmission is initiated by any instruction that uses SBUF as a destination register. The "write to SBUF" signal also loads a 1 into the 9th bit position of the transmit shift register and flags the TX Control unit that a transmission is requested. Transmission actually commences at S1P1 of the machine cycle following the next rollover in the divide-by-16 counter. (Thus, the bit times are synchronized to the divide-by-16 counter, not to the "write to SBUF" signal.)

The transmission begins with activation of  $\overline{\text{SEND}}$ , which puts the start bit at TXD. One bit time later, DATA is activated, which enables the output bit of the transmit shift register to TXD. The first shift pulse occurs one bit time after that.

As data bits shift out to the right, zeros are clocked in from the left. When the MSB of the data byte is at the output position of the shift register, then the 1 that was initially loaded into the 9th position is just to the left of the MSB, and all positions to the left of that contain zeroes. This condition flags the TX Control unit to do one last shift and then deactivate  $\overline{\text{SEND}}$  and set TI. This occurs at the 10th divide-by-16 rollover after "write to SBUF."

Reception is initiated by a detected 1-to-0 transition at RXD. For this purpose RXD is sampled at a rate of 16 times whatever baud rate has been established. When a transition is detected, the divide-by-16 counter is immediately reset, and 1FFH is written into the input shift register. Resetting the divide-by-16 counter aligns its rollovers with the boundaries of the incoming bit times.

The 16 states of the counter divide each bit time into 16ths. At the 7th, 8th, and 9th counter states of each bit time, the bit detector samples the value of RXD. The value accepted is the value that was seen in at least 2 of the 3 samples. This is done for noise rejection. If the value accepted during the first bit time is not 0, the receive circuits are reset and the unit goes back to looking for another 1-to-0 transition. This is to provide rejection of false start bits. If the start bit proves valid, it is shifted into the input shift register, and reception of the rest of the frame will proceed.

As data bits come in from the right, 1s shift out to the left. When the start bit arrives at the leftmost position in the shift register, (which in mode 1 is a 9-bit register), it flags the RX Control block to do one last shift, load SBUF and RB8, and set RI. The signal to load SBUF and RB8, and to set RI, will be generated if, and only if, the following conditions are met at the time the final shift pulse is generated.

- 1) RI = 0, and
- 2) Either SM2 = 0, or the received stop bit = 1

If either of these two conditions is not met, the received frame is irretrievably lost. If both conditions are met, the stop bit goes into RB8, the 8 data bits go into SBUF, and RI is activated. At this time, whether the above conditions

are met or not, the unit goes back to looking for a 1-to-0 transition in RXD.

## 7.7.6 More About Modes 2 and 3

Eleven bits are transmitted (through TXD), or received (through RXD): a start bit (0), 8 data bits (LSB first), a programmable 9th data bit, and a stop bit (1). On transmit, the 9th data bit (TB8) can be assigned the value of 0 or 1. On receive, the 9th data bit goes into RB8 in SCON. The baud rate is programmable to either 1/32 or 1/64 the oscillator frequency in mode 2. Mode 3 may have a variable baud rate generated from either Timer 1 or 2 depending on the state of TCLK and RCLK.

Figures 7-21 A and B show a functional diagram of the serial port in modes 2 and 3. The receive portion is exactly the same as in mode 1. The transmit portion differs from mode 1 only in the 9th bit of the transmit shift register.

Transmission is initiated by any instruction that uses SBUF as a destination register. The "write to SBUF" signal also loads TB8 into the 9th bit position of the transmit shift register and flags the TX Control unit that a transmission is requested. Transmission commences at S1P1 of the machine cycle following the next rollover in the divide-by-16 counter. (Thus, the bit times are synchronized to the divide-by-16 counter, not to the "write to SBUF" signal.)

The transmission begins with activation of  $\overline{\text{SEND}}$ , which puts the start bit at TXD. One bit time later, DATA is activated, which enables the output bit of the transmit shift register to TXD. The first shift pulse occurs one bit time after that. The first shift clocks a 1 (the stop bit) into the 9th bit position of the shift register. Thereafter, only zeroes are clocked in. Thus, as data bits shift out to the right, zeroes are clocked in from the left. When TB8 is at the output position of the shift register, then the stop bit is just to the left of TB8, and all positions to the left of that contain zeroes. This condition flags the TX Control unit to do one last shift and then deactivate  $\overline{\text{SEND}}$  and set TI. This occurs at the 11th divide-by-16 rollover after "write to SBUF."

Reception is initiated by a detected 1-to-0 transition at RXD. For this purpose RXD is sampled at a rate of 16 times whatever baud rate has been established. When a transition is detected, the divide-by-16 counter is immediately reset, and 1FFH is written to the input shift register.

At the 7th, 8th and 9th counter states of each bit time, the bit detector samples the value of RXD. The value accepted is the value that was seen in at least 2 of the 3 samples. If the value accepted during the first bit time is not 0, the receive circuits are reset and the unit goes back to looking for another 1-to-0 transition. If the start bit proves valid, it is shifted into the input shift register, and reception of the rest of the frame will proceed.

# MCS<sup>®</sup>-51 ARCHITECTURE

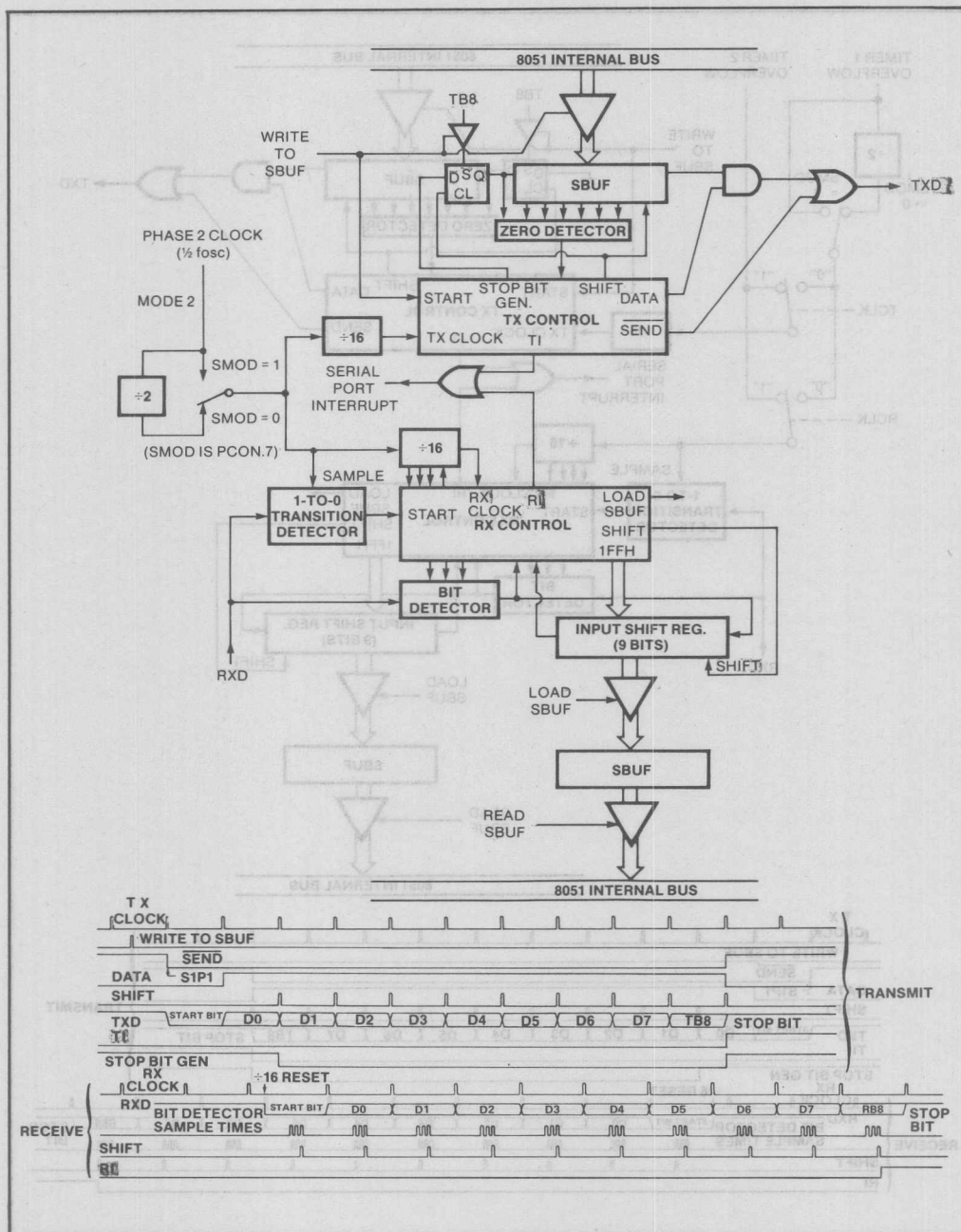


Figure 7-21A. Serial Port Mode 2

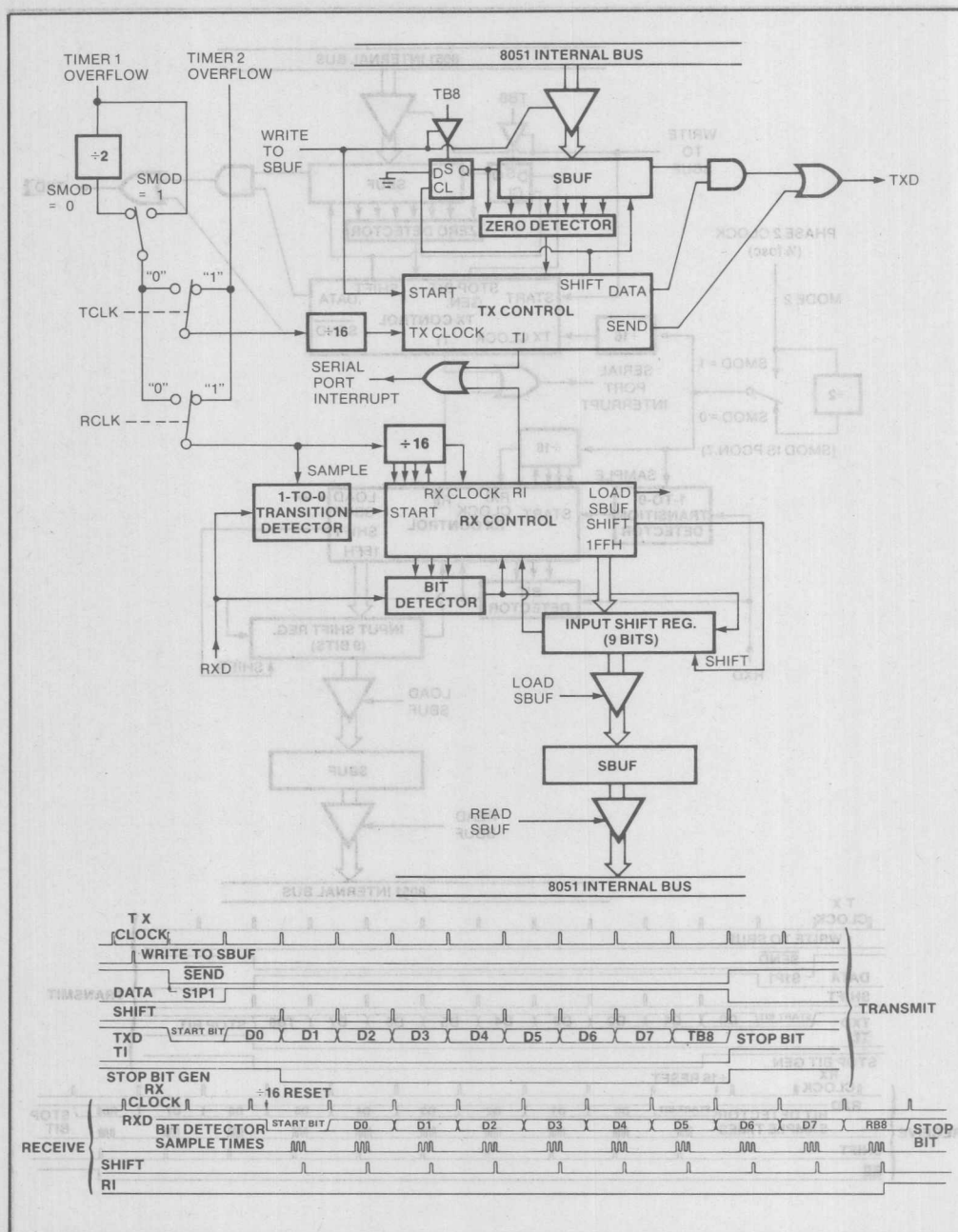


Figure 7-21B. Serial Port Mode 3  
TCLK, RCLK, and Timer 2 are present in the 8052/8032 only.



As data bits come in from the right, 1s shift out to the left. When the start bit arrives at the leftmost position in the shift register (which in modes 2 and 3 is a 9-bit register), it flags the RX Control block to do one last shift, load SBUF and RB8, and set RI. The signal to load SBUF and RB8, and to set RI, will be generated if, and only if, the following conditions are met at the time the final shift pulse is generated:

- 1) RI = 0, and
- 2) Either SM2 = 0 or the received 9th data bit = 1

If either of these conditions is not met, the received frame is irretrievably lost, and RI is not set. If both conditions are met, the received 9th data bit goes into RB8, and the first 8 data bits go into SBUF. One bit time later, whether the above conditions were met or not, the unit goes back to looking for a 1-to-0 transition at the RXD input.

Note that the value of the received stop bit is irrelevant to SBUF, RB8, or RI.

## 7.8 INTERRUPTS

The 8051 provides 5 interrupt sources. The 8052 provides 6. These are shown in Figure 7-22.

The External Interrupts  $\overline{\text{INT0}}$  and  $\overline{\text{INT1}}$  can each be either level-activated or transition-activated, depending on bits

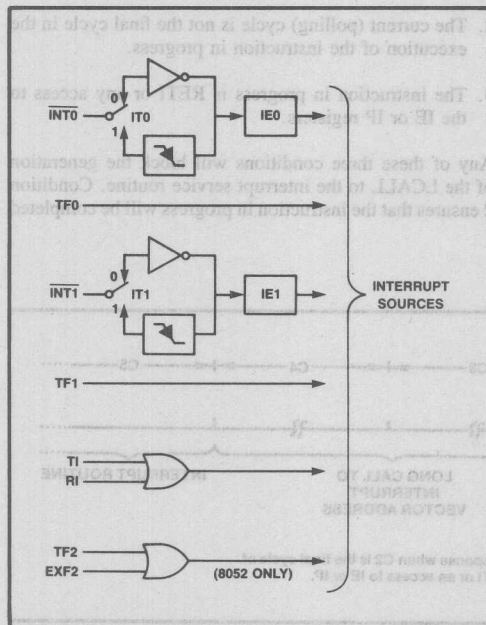


Figure 7-22. MCS-51 Interrupt Sources

IT0 and IT1 in Register TCON. The flags that actually generate these interrupts are bits IE0 and IE1 in TCON. When an external interrupt is generated, the flag that generated it is cleared by the hardware when the service routine is vectored to only if the interrupt was transition-activated. If the interrupt was level-activated, then the external requesting source is what controls the request flag, rather than the on-chip hardware.

The Timer 0 and Timer 1 Interrupts are generated by TF0 and TF1, which are set by a rollover in their respective timer/counter registers (except see Section 7.6.1 for Timer 0 in mode 3). When a timer interrupt is generated, the flag that generated it is cleared by the on-chip hardware when the service routine is vectored to.

The Serial Port Interrupt is generated by the logical OR of RI and TI. Neither of these flags is cleared by hardware when the service routine is vectored to. In fact, the service routine will normally have to determine whether it was RI or TI that generated the interrupt, and the bit will have to be cleared in software.

In the 8052, the Timer 2 Interrupt is generated by the logical OR of TF2 and EXF2. Neither of these flags is cleared by hardware when the service routine is vectored to. In fact, the service routine may have to determine whether it was TF2 or EXF2 that generated the interrupt, and the bit will have to be cleared in software.

All of the bits that generate interrupts can be set or cleared by software, with the same result as though it had been set or cleared by hardware. That is, interrupts can be generated or pending interrupts can be canceled in software.

| (MSB)  |          |  |    | (LSB) |     |     |     |
|--------|----------|--|----|-------|-----|-----|-----|
| EA     | X        | ET2  | ES | ET1   | EX1 | ET0 | EX0 |
| Symbol | Position | Function   |    |       |     |     |     |
| EA     | IE.7     | disables all interrupts. If EA = 0, no interrupt will be acknowledged. If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit. |    |       |     |     |     |
| —      | IE.6     | reserved   |    |       |     |     |     |
| ET2    | IE.5     | enables or disables the Timer 2 overflow or capture interrupt. If ET2 = 0, the Timer 2 interrupt is disabled.  |    |       |     |     |     |
| ES     | IE.4     | enables or disables the Serial Port interrupt. If ES = 0, the Serial Port interrupt is disabled.   |    |       |     |     |     |
| ET1    | IE.3     | enables or disables the Timer 1 Overflow interrupt. If ET1 = 0, the Timer 1 interrupt is disabled.   |    |       |     |     |     |
| EX1    | IE.2     | enables or disables External Interrupt 1. If EX1 = 0, External Interrupt 1 is disabled.  |    |       |     |     |     |
| ET0    | IE.1     | enables or disables the Timer 0 Overflow interrupt. If ET0 = 0, the Timer 0 interrupt is disabled.   |    |       |     |     |     |
| EX0    | IE.0     | enables or disables External Interrupt 0. If EX0 = 0, External Interrupt 0 is disabled.  |    |       |     |     |     |

Figure 7-23. IE: Interrupt Enable Register

Each of these interrupt sources can be individually enabled or disabled by setting or clearing a bit in Special Function Register IE (Figure 7-23). Note that IE contains also a global disable bit, EA, which disables all interrupts at once.

### 7.8.1 Priority Level Structure

Each interrupt source can also be individually programmed to one of two priority levels by setting or clearing a bit in Special Function Register IP (Figure 7-24). A low-priority interrupt can itself be interrupted by a high-priority interrupt, but not by another low-priority interrupt. A high-priority interrupt can't be interrupted by any other interrupt source.

| (MSB)  |          |  |    | (LSB) |     |     |     |
|--------|----------|--|----|-------|-----|-----|-----|
| X      | X        | PT2  | PS | PT1   | PX1 | PT0 | PX0 |
| Symbol | Position | Function   |    |       |     |     |     |
| —      | IP.7     | reserved   |    |       |     |     |     |
| —      | IP.6     | reserved   |    |       |     |     |     |
| PT2    | IP.5     | defines the Timer 2 interrupt priority level. PT2 = 1 programs it to the higher priority level.    |    |       |     |     |     |
| PS     | IP.4     | defines the Serial Port interrupt priority level. PS = 1 programs it to the higher priority level. |    |       |     |     |     |
| PT1    | IP.3     | defines the Timer 1 interrupt priority level. PT1 = 1 programs it to the higher priority level.    |    |       |     |     |     |
| PX1    | IP.2     | defines the External Interrupt 1 priority level. PX1 = 1 programs it to the higher priority level. |    |       |     |     |     |
| PT0    | IP.1     | defines the Timer 0 interrupt priority level. PT0 = 1 programs it to the higher priority level.    |    |       |     |     |     |
| PX0    | IP.0     | defines the External Interrupt 0 priority level. PX0 = 1 programs it to the higher priority level. |    |       |     |     |     |

Figure 7-24. IP: Interrupt Priority Register

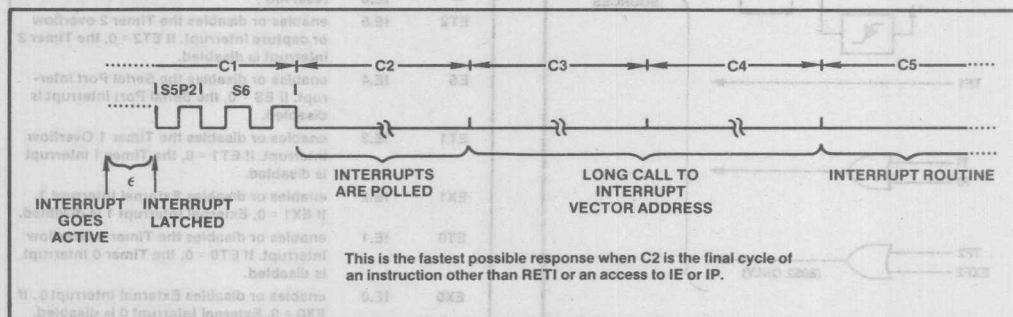


Figure 7-25. Interrupt Response Timing Diagram

If two requests of different priority levels are received simultaneously, the request of higher priority level is serviced. If requests of the **same** priority level are received simultaneously, an internal polling sequence determines which request is serviced. Thus within each priority level there is a second priority structure determined by the polling sequence, as follows:

| SOURCE        | PRIORITY WITHIN LEVEL |
|---------------|-----------------------|
| 1. IE0        | (highest)             |
| 2. TF0        |                       |
| 3. IE1        |                       |
| 4. TF1        |                       |
| 5. RI + TI    |                       |
| 6. TF2 + EXF2 | (lowest)              |

Note that the "priority within level" structure is only used to resolve simultaneous requests of the same priority level.

### 7.8.2 How Interrupts Are Handled

The interrupt flags are sampled at S5P2 of every machine cycle. The samples are polled during the following machine cycle. If one of the flags was in a set condition at S5P2 of the preceding cycle, the polling cycle will find it and the interrupt system will generate an LCALL to the appropriate service routine, provided this hardware-generated LCALL is not blocked by any of the following conditions:

1. An interrupt of equal or higher priority level is already in progress.
2. The current (polling) cycle is not the final cycle in the execution of the instruction in progress.
3. The instruction in progress is RETI or any access to the IE or IP registers.

Any of these three conditions will block the generation of the LCALL to the interrupt service routine. Condition 2 ensures that the instruction in progress will be completed

before vectoring to any service routine. Condition 3 ensures that if the instruction in progress is RETI or any access to IE or IP, then at least *one more* instruction will be executed before any interrupt is vectored to.

The polling cycle is repeated with each machine cycle, and the values polled are the values that were present at S5P2 of the previous machine cycle. Note then that if an interrupt flag is active but not being responded to for one of the above conditions, if the flag is not *still* active when the blocking condition is removed, the denied interrupt will not be serviced. In other words, the fact that the interrupt flag was once active but not serviced is not remembered. Every polling cycle is new.

The polling cycle/LCALL sequence is illustrated in Figure 7-25.

Note that if an interrupt of higher priority level goes active prior to S5P2 of the machine cycle labeled C3 in Figure 7-25, then in accordance with the above rules it will be vectored to during C5 and C6, without any instruction of the lower priority routine having been executed.

Thus the processor acknowledges an interrupt request by executing a hardware-generated LCALL to the appropriate servicing routine. In some cases it also clears the flag that generated the interrupt, and in other cases it doesn't. It never clears the Serial Port or Timer 2 flags. This has to be done in the user's software. It clears an external interrupt flag (IE0 or IE1) only if it was transition-activated. The hardware-generated LCALL pushes the contents of the Program Counter onto the stack (but it does not save the PSW) and reloads the PC with an address that depends on the source of the interrupt being vectored to, as shown below:

| SOURCE     | VECTOR ADDRESS |
|------------|----------------|
| IE0        | 0003H          |
| TF0        | 000BH          |
| IE1        | 0013H          |
| TF1        | 001BH          |
| RI + TI    | 0023H          |
| TF2 + EXF2 | 002BH          |

Execution proceeds from that location until the RETI instruction is encountered. The RETI instruction informs the processor that this interrupt routine is no longer in progress, then pops the top two bytes from the stack and reloads the Program Counter. Execution of the interrupted program continues from where it left off.

Note that a simple RET instruction would also have returned execution to the interrupted program, but it would have left the interrupt control system thinking an interrupt was still in progress.

### 7.8.3 External Interrupts

The external sources can be programmed to be level-activated or transition-activated by setting or clearing bit IT1 or IT0 in Register TCON. If  $ITx = 0$ , external interrupt  $x$  is triggered by a detected low at the  $\overline{INTx}$  pin. If  $ITx = 1$ , external interrupt  $x$  is edge-triggered. In this mode if successive samples of the  $\overline{INTx}$  pin show a high in one cycle and a low in the next cycle, interrupt request flag IEx in TCON is set. Flag bit IEx then requests the interrupt.

Since the external interrupt pins are sampled once each machine cycle, an input high or low should hold for at least 12 oscillator periods to ensure sampling. If the external interrupt is transition-activated, the external source has to hold the request pin high for at least one cycle, and then hold it low for at least one cycle to ensure that the transition is seen so that interrupt request flag IEx will be set. IEx will be automatically cleared by the CPU when the service routine is called.

If the external interrupt is level-activated, the external source has to hold the request active until the requested interrupt is actually generated. Then it has to deactivate the request before the interrupt service routine is completed, or else another interrupt will be generated.

### 7.8.4 Response Time

The  $\overline{INT0}$  and  $\overline{INT1}$  levels are inverted and latched into IE0 and IE1 at S5P2 of every machine cycle. The values are not actually polled by the circuitry until the next machine cycle. If a request is active and conditions are right for it to be acknowledged, a hardware subroutine call to the requested service routine will be the next instruction to be executed. The call itself takes two cycles. Thus, a minimum of three complete machine cycles elapse between activation of an external interrupt request and the beginning of execution of the first instruction of the service routine. Figure 7-25 shows interrupt response timings.

A longer response time would result if the request is blocked by one of the 3 previously listed conditions. If an interrupt of equal or higher priority level is already in progress, the additional wait time obviously depends on the nature of the other interrupt's service routine. If the instruction in progress is not in its final cycle, the additional wait time cannot be more than 3 cycles, since the longest instructions (MUL and DIV) are only 4 cycles long, and if the instruction in progress is RETI or an access to IE or IP, the additional wait time cannot be more than 5 cycles (a maximum of one more cycle to complete the instruction in progress, plus 4 cycles to complete the next instruction if the instruction is MUL or DIV).

Thus, in a single-interrupt system, the response time is always more than 3 cycles and less than 9 cycles.

## 7.9 SINGLE-STEP OPERATION

The 8051 interrupt structure allows single-step execution with very little software overhead. As previously noted, an interrupt request will not be responded to while an interrupt of equal priority level is still in progress, nor will it be responded to after RETI until at least one other instruction has been executed. Thus, once an interrupt routine has been entered, it cannot be re-entered until at least one instruction of the interrupted program is executed. One way to use this feature for single-stop operation is to program one of the external interrupts (say, INT0) to be level-activated. The service routine for the interrupt will terminate with the following code:

```
JNB P3.2,$ ;WAIT HERE TILL INTO
                GOES HIGH
JB P3.2,$ ;NOW WAIT HERE TILL
                IT GOES LOW
RETI ;GO BACK AND
                EXECUTE ONE
                INSTRUCTION
```

Now if the INT0 pin, which is also the P3.2 pin, is held normally low, the CPU will go right into the External Interrupt 0 routine and stay there until INT0 is pulsed (from low to high to low). Then it will execute RETI, go back to the task program, execute one instruction, and immediately re-enter the External Interrupt 0 routine to await the next pulsing of P3.2. One step of the task program is executed each time P3.2 is pulsed.

## 7.10 RESET

The reset input is the RST pin, which is the input to a Schmitt Trigger.

A reset is accomplished by holding the RST pin high for at least two machine cycles (24 oscillator periods), while the oscillator is running. The CPU responds by executing

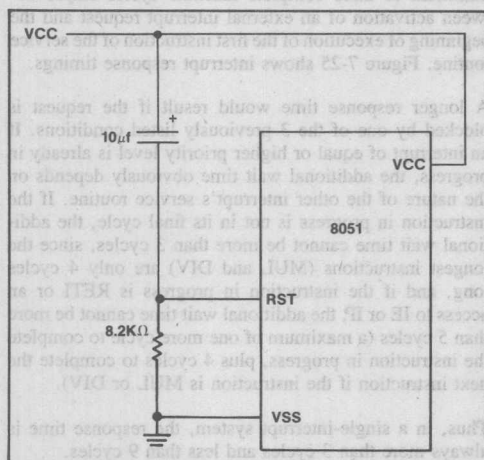


Figure 7-26. Power on Reset Circuit

an internal reset. It also configures the ALE and PSEN pins as inputs. (They are quasi-bidirectional). The internal reset is executed during the second cycle in which RST is high and is repeated every cycle until RST goes low. It leaves the internal registers as follows:

| REGISTER           | CONTENT       |
|--------------------|---------------|
| PC                 | 0000H         |
| ACC                | 00H           |
| B                  | 00H           |
| PSW                | 00H           |
| SP                 | 07H           |
| DPTR               | 0000H         |
| P0-P3              | 0FFH          |
| IP (8051)          | XXX00000B     |
| IP (8052)          | XX000000B     |
| IE (8051)          | 0XX00000B     |
| IE (8052)          | 0X000000B     |
| TMOD               | 00H           |
| TCON               | 00H           |
| T2CON (8052 only)  | 00H           |
| TH0                | 00H           |
| TL0                | 00H           |
| TH1                | 00H           |
| TL1                | 00H           |
| TH2                | 00H           |
| TL2                | 00H           |
| RCAP2H (8052 only) | 00H           |
| RCAP2L (8052 only) | 00H           |
| SCON               | 00H           |
| SBUF               | Indeterminate |
| PCON (HMOS)        | 0XXXXXXB      |
| PCON (CHMOS)       | 0XXXX000B     |

The internal RAM is not affected by reset. When VCC is turned on, the RAM content is indeterminate unless the part is returning from a reduced power mode of operation.

## POWER-ON RESET

An automatic reset can be obtained when VCC is turned on by connecting the RST pin to VCC through a 10 µF capacitor and to VSS through an 8.2KΩ resistor, providing the VCC risetime does not exceed a millisecond and the oscillator start-up time does not exceed 10 milliseconds. This power-on reset circuit is shown in Figure 7-26. When power comes on, the current drawn by RST commences to charge the capacitor. The voltage at RST is the difference between VCC and the capacitor voltage, and decreases from VCC as the cap charges. The larger the capacitor, the more slowly VRST decreases. VRST must remain above the lower threshold of the Schmitt Trigger long enough to effect a complete reset. The time required is the oscillator start-up time, plus 2 machine cycles.

## 7.11 POWER-SAVING MODES OF OPERATION

For applications where power consumption is critical the CHMOS version provides power reduced modes of operation as a standard feature. The power down mode in



HMOS devices is no longer a standard feature and is being phased out.

### 7.11.1 CHMOS Power Reduction Modes

CHMOS versions have two power-reducing modes, Idle and Power Down. The input through which backup power is supplied during these operations is VCC. Figure 7-27 shows the internal circuitry which implements these features. In the Idle mode (IDL = 1), the oscillator continues to run and the Interrupt, Serial Port, and Timer blocks continue to be clocked, but the clock signal is gated off to the CPU. In Power Down (PD = 1), the oscillator is frozen. The Idle and Power Down modes are activated by setting bits in Special Function Register PCON. The address of this register is 87H. Figure 7-28 details its contents.

#### IDLE MODE

An instruction that sets PCON.0 causes that to be the last instruction executed before going into the Idle mode. In the Idle mode, the internal clock signal is gated off to the CPU, but not to the Interrupt, Timer, and Serial Port functions. The CPU status is preserved in its entirety: the Stack Pointer, Program Counter, Program Status Word, Accumulator, and all other registers maintain their data during Idle. The port pins hold the logical states they had at the time Idle was activated. ALE and PSEN hold at logic high levels.

There are two ways to terminate the Idle. Activation of any enabled interrupt will cause PCON.0 to be cleared by hardware, terminating the Idle mode. The interrupt will be serviced, and following RETI the next instruction to

| (MSB) |   |   |   | (LSB) |     |    |     |
|-------|---|---|---|-------|-----|----|-----|
| SMOD  | — | — | — | GF1   | GF0 | PD | IDL |

| Symbol | Position | Name and Function   |
|--------|----------|---|
| SMOD   | PCON.7   | Double Baud rate bit. When set to a 1 and Timer 1 is used to generate baud rate, and the Serial Port is used in modes 1, 2, or 3. |
| —      | PCON.6   | (Reserved)  |
| —      | PCON.5   | (Reserved)  |
| —      | PCON.4   | (Reserved)  |
| GF1    | PCON.3   | General-purpose flag bit.   |
| GF0    | PCON.2   | General-purpose flag bit.   |
| PD     | PCON.1   | Power Down bit. Setting this bit activates power down operation.  |
| IDL    | PCON.0   | Idle mode bit. Setting this bit activates idle mode operation.  |

If 1s are written to PD and IDL at the same time, PD takes precedence. The reset value of PCON is (0XXX0000).

Figure 7-28. PCON: Power Control Register

be executed will be the one following the instruction that put the device into Idle.

The flag bits GF0 and GF1 can be used to give an indication if an interrupt occurred during normal operation or during an Idle. For example, an instruction that activates Idle can also set one or both flag bits. When Idle is terminated by an interrupt, the interrupt service routine can examine the flag bits.

The other way of terminating the Idle mode is with a hardware reset. Since the clock oscillator is still running, the hardware reset needs to be held active for only two machine cycles (24 oscillator periods) to complete the

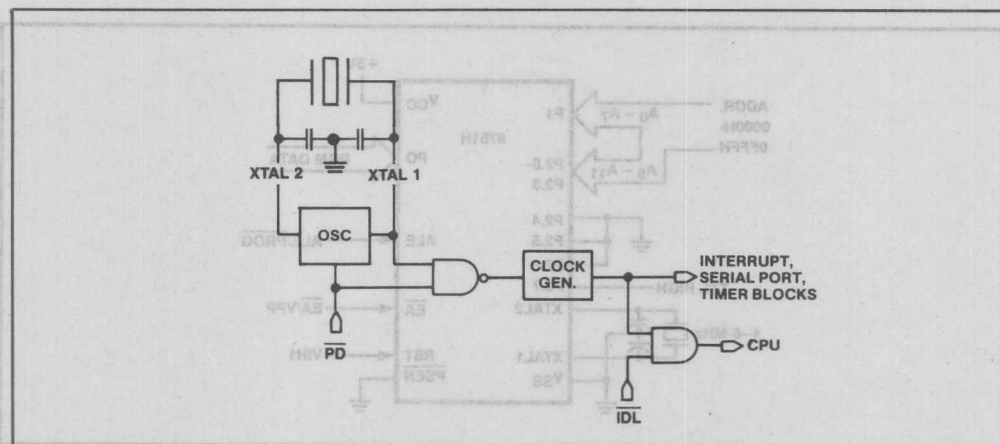


Figure 7-27. Idle and Power Down Hardware

reset.

## POWER DOWN MODE

An instruction that sets PCON.1 causes that to be the last instruction executed before going into the Power Down mode. In the Power Down mode, the on-chip oscillator is stopped. With the clock frozen, all functions are stopped, but the on-chip RAM and Special Function Registers are held. The port pins output the values held by their respective SFRs. ALE and PSEN output lows.

The only exit from Power Down is a hardware reset. Reset redefines all the SFRs, but does not change the on-chip RAM.

In the Power down mode of operation, VCC can be reduced to minimize power consumption. Care must be taken, however, to ensure that VCC is not reduced before the Power Down mode is invoked, and that VCC is restored to its normal operating level, before the Power Down mode is terminated. The reset that terminates Power Down also frees the oscillator. The reset should not be activated before VCC is restored to its normal operating level, and must be held active long enough to allow the oscillator to restart and stabilize (normally less than 10 msec).

## 7.12 8751H

The 8751H is the EPROM member of the MCS-51 family. This means that the on-chip Program Memory can be electrically programmed, and can be erased by exposure to ultraviolet light. The 8751H also has a provision for

denying external access to the on-chip Program Memory, in order to protect its contents against software piracy.

### 7.12.1 Programming the EPROM

To be programmed, the 8751H must be running with a 4 to 6 MHz oscillator. (The reason the oscillator needs to be running is that the internal bus is being used to transfer address and program data to appropriate internal registers.) The address of an EPROM location to be programmed is applied to Port 1 and pins P2.0–P2.3 of Port 2, while the data byte is applied to Port 0. Pins P2.4–P2.6 and  $\overline{\text{PSEN}}$  should be held low, and P2.7 and RST high. (These are all TTL levels except RST, which requires 2.5V for a logic high.)  $\overline{\text{EA}}/\text{VPP}$  is held normally high, and is pulsed to +21V. While  $\overline{\text{EA}}/\text{VPP}$  is at 21V, the ALE/PROG pin, which is normally being held high, is pulsed low for 50 msec. Then  $\overline{\text{EA}}/\text{VPP}$  is returned to high. This setup is shown in Figure 7.29. Detailed timing specifications are provided in the 8751H data sheet.

Note: The  $\overline{\text{EA}}$  pin must not be allowed to go above the maximum specified VPP level of 21.5V for any amount of time. Even a narrow glitch above that voltage level can cause permanent damage to the device. The VPP source should be well regulated and free of glitches.

### 7.12.2 Program Verification

If the program security bit has not been programmed, the on-chip Program Memory can be read out for verification purposes, if desired, either during or after the programming operation. The required setup, which is shown in Figure 7.30, is the same as for programming the EPROM

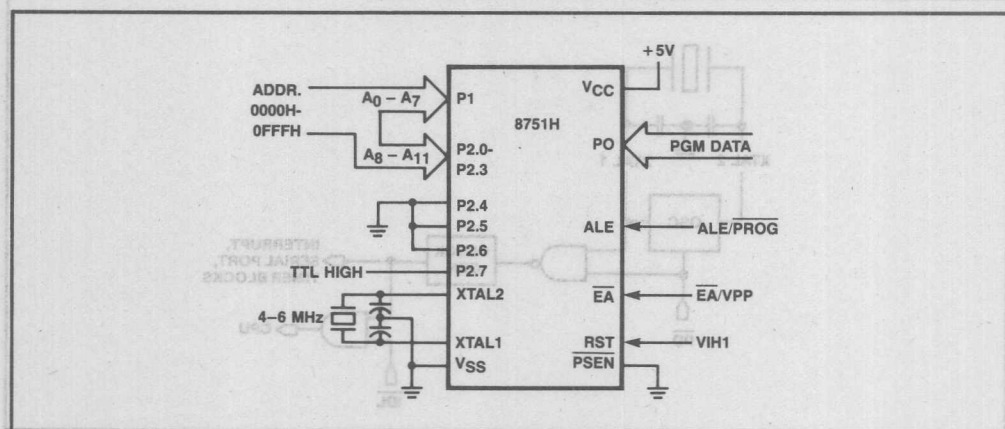


Figure 7-29. Programming the 8751H

except that pin P2.7 is held at TTL low (or used as an active-low read strobe). The address of the Program Memory location to be read is applied to Port 1 and pins P2.0–P2.3. The other Port 2 pins and PSEN are held low. ALE,  $\overline{\text{EA}}$ , and RST are held high. The contents of the addressed location will come out on Port 0. External pull-ups are required on Port 0 for this operation.

### 7.12.3 Program Memory Security

The 8751H contains a security bit, which, once programmed, denies electrical access by any external means to the on-chip Program Memory. The setup and procedure for programming the security bit are the same as for normal programming, except that pin P2.6 is held at TTL high.

The setup is shown in Figure 7.31. Port 0, Port 1, and pins P2.0–P2.3 of Port 2 may be in any state.

Once the security bit has been programmed, it can be deactivated only by full erasure of the Program Memory. While it is programmed, the internal Program Memory cannot be read out, the device cannot be further programmed, and it cannot execute external program memory. Erasing the EPROM, thus deactivating the security bit, restores the device's full functionality. It can then be re-programmed.

### 7.12.4 Erasure Characteristics

Erasure of the 8751H Program Memory begins to occur when the chip is exposed to light with wavelengths shorter

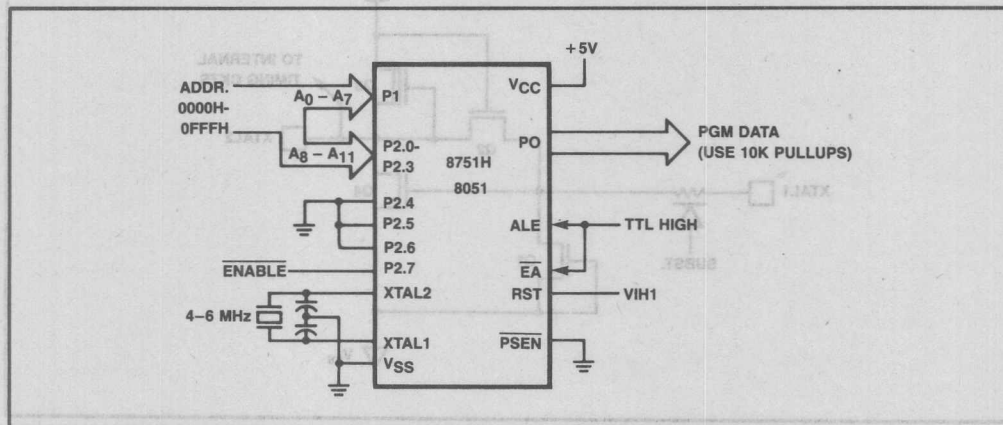


Figure 7-30. Program Verification in the 8751H and 8051

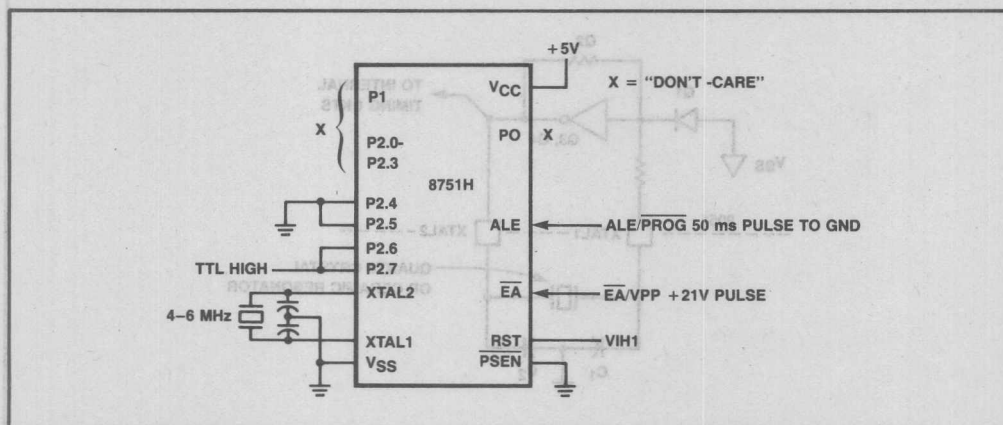


Figure 7-31. Programming the Security Bit in the 8751H

than approximately 4,000 Angstroms. Since sunlight and fluorescent lighting have wavelengths in this range, exposure to these light sources over an extended time (about 1 week in sunlight, or 3 years in room-level fluorescent lighting) could cause inadvertent erasure. If an application subjects the 8751H to this type of exposure, it is suggested that an opaque label be placed over the window.

The recommended erasure procedure is exposure to ultraviolet light (at 2537 Angstroms) to an integrated dose of at least 15 W/cm<sup>2</sup>. Exposing the 8751H to an ultraviolet lamp of 12,000  $\mu$ W/cm<sup>2</sup> rating for 20 to 30 minutes, at a distance of about 1 inch, should be sufficient.

## 7.13 MORE ABOUT THE ON-CHIP OSCILLATOR

### 7.13.1 HMOS Versions

The on-chip oscillator circuitry for the HMOS (HMOS-I and HMOS-II) members of the MCS-51 family is a single stage linear inverter (Figure 7-32), intended for use as a crystal-controlled, positive reactance oscillator (Figure 7-33). In this application the crystal is operated in its fundamental response mode as an inductive reactance in parallel resonance with capacitance external to the crystal.

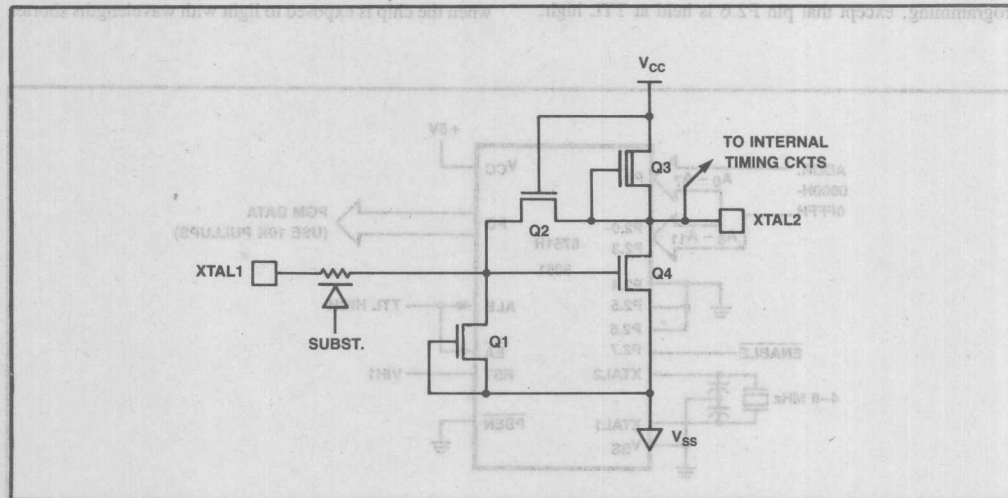


Figure 7-32. On-Chip Oscillator Circuitry in the HMOS Versions of the MCS-51 Family

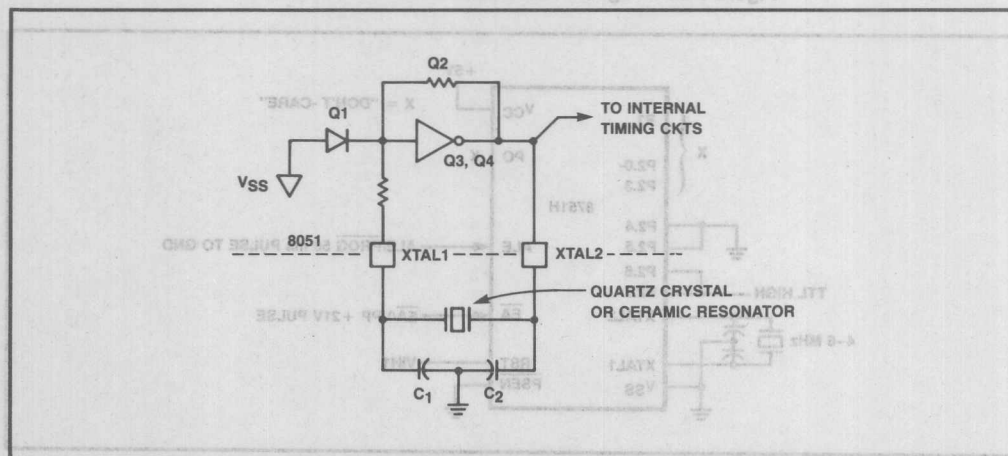


Figure 7-33. Using the HMOS On-Chip Oscillator



The crystal specifications and capacitance values (C1 and C2 in Figure 7-33) are not critical. 30 pF can be used in these positions at any frequency with good quality crystals. A ceramic resonator can be used in place of the crystal in cost-sensitive applications. When a ceramic resonator is used, C1 and C2 are normally selected to be of somewhat higher values, typically, 47 pF. The manufacturer of the ceramic resonator should be consulted for recommendations on the values of these capacitors.

A more in-depth discussion of crystal specifications, ce-

ramic resonators, and the selection of values for C1 and C2 can be found in Application Note AP-155, "Oscillators for Microcontrollers," which is included in this manual.

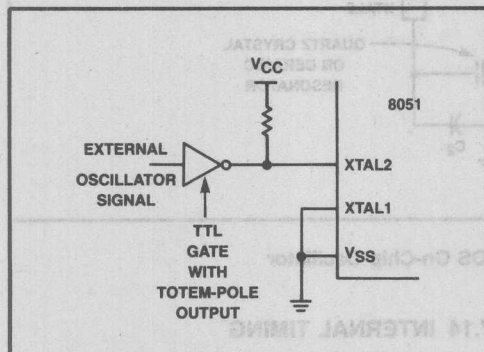
To drive the HMOS parts with an external clock source, apply the external clock signal to XTAL2, and ground XTAL1, as shown in Figure 7-34. A pull-up resistor may be used (to increase noise margin), but is optional if VOH of the driving gate exceeds the VIHMIN specification of XTAL2.

### 7.13.2 CHMOS

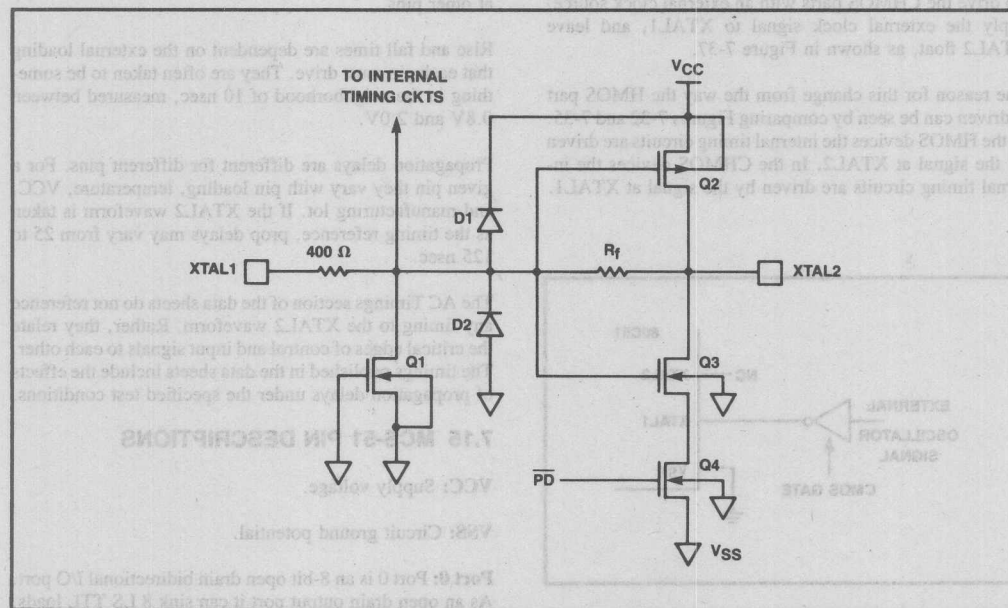
The on-chip oscillator circuitry for the 80C51, shown in Figure 7-35, consists of a single stage linear inverter intended for use as a crystal-controlled, positive reactance oscillator in the same manner as the HMOS parts. However, there are some important differences.

One difference is that the 80C51 is able to turn off its oscillator under software control (by writing a 1 to the PD bit in PCON). Another difference is that in the 80C51 the internal clocking circuitry is driven by the signal at XTAL1, whereas in the HMOS versions it is by the signal at XTAL2.

The feedback resistor  $R_f$  in Figure 7-35 consists of paralleled n- and p-channel FETs controlled by the PD bit, such that  $R_f$  is opened when PD = 1. The diodes D1 and D2, which act as clamps to VCC and VSS, are parasitic to the  $R_f$  FETs.



**Figure 7-34. Driving the HMOS MCS-51 Parts with an External Clock Source**



**Figure 7-35. On-Chip Oscillator Circuitry in the CHMOS Versions of the MCS-51 Family**

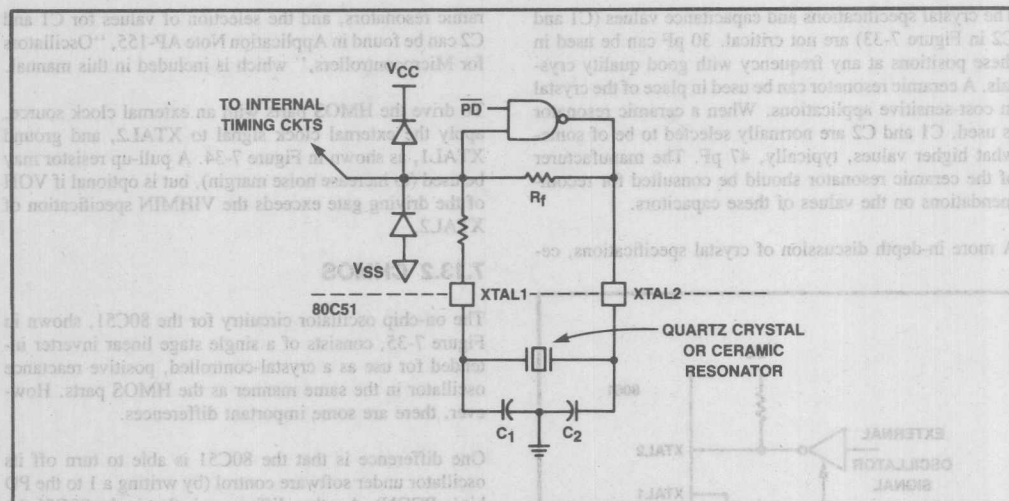


Figure 7-36. Using the CHMOS On-Chip Oscillator

The oscillator can be used with the same external components as the HMOS versions, as shown in Figure 7-36. Typically,  $C_1 = C_2 = 30$  pF when the feedback element is a quartz crystal, and  $C_1 = C_2 = 47$  pF when a ceramic resonator is used.

To drive the CHMOS parts with an external clock source, apply the external clock signal to XTAL1, and leave XTAL2 float, as shown in Figure 7-37.

The reason for this change from the way the HMOS part is driven can be seen by comparing Figures 7-32 and 7-35. In the HMOS devices the internal timing circuits are driven by the signal at XTAL2. In the CHMOS devices the internal timing circuits are driven by the signal at XTAL1.

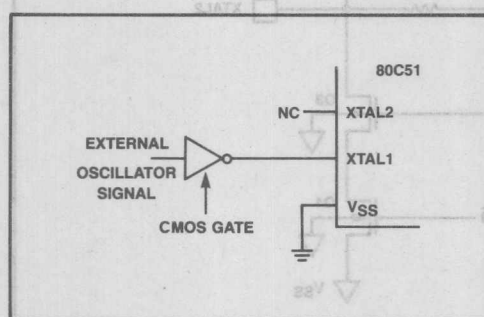


Figure 7-37. Driving the CHMOS MCS-51 Parts with an External Clock Source

## 7.14 INTERNAL TIMING

Figures 7-38 through 7-41 show when the various strobe and port signals are clocked internally. The figures do not show rise and fall times of the signals, nor do they show propagation delays between the XTAL2 signal and events at other pins.

Rise and fall times are dependent on the external loading that each pin must drive. They are often taken to be something in the neighborhood of 10 nsec, measured between 0.8V and 2.0V.

Propagation delays are different for different pins. For a given pin they vary with pin loading, temperature, VCC, and manufacturing lot. If the XTAL2 waveform is taken as the timing reference, prop delays may vary from 25 to 125 nsec.

The AC Timings section of the data sheets do not reference any timing to the XTAL2 waveform. Rather, they relate the critical edges of control and input signals to each other. The timings published in the data sheets include the effects of propagation delays under the specified test conditions.

## 7.15 MCS-51 PIN DESCRIPTIONS

VCC: Supply voltage.

VSS: Circuit ground potential.

**Port 0:** Port 0 is an 8-bit open drain bidirectional I/O port. As an open drain output port it can sink 8 LS TTL loads. Port 0 pins that have 1s written to them float, and in that state will function as high-impedance inputs. Port 0 is also

the multiplexed low-order address and data bus during accesses to external memory. In this application it uses strong internal pullups when emitting 1s. Port 0 also emits code bytes during program verification. In that application, external pullups are required.

**Port 1:** Port 1 is an 8-bit bidirectional I/O port with internal pullups. The port 1 output buffers can sink/source 4 LS TTL loads. Port 1 pins that have 1s written to them are pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 1 pins that are externally being pulled low will source current (ILL, on the data sheet) because of the internal pullups.

In the 8052, pins P1.0 and P1.1 also serve the alternate functions of T2 and T2EX. T2 is the Timer 2 external input. T2EX is the input through which a Timer 2 "capture" is triggered.

**Port 2:** Port 2 is an 8-bit bidirectional I/O port with internal pullups. The Port 2 output buffers can sink/source 4 LS TTL loads. Port 2 emits the high-order address byte during accesses to external memory that use 16-bit addresses. In this application it uses the strong internal pullups when emitting 1s. Port 2 also receives the high-order address and control bits during 8751H programming and verification, and during program verification in the 8051AH.

**Port 3:** Port 3 is an 8-bit bidirectional I/O port with internal pullups. It also serves the functions of various special features of the MCS-51 Family, as listed below:

| PORT PIN | ALTERNATE FUNCTION                     |
|----------|--|
| P3.0     | RXD (serial input port)                |
| P3.1     | TXD (serial output port)               |
| P3.2     | INT0 (external interrupt 0)            |
| P3.3     | INT1 (external interrupt 1)            |
| P3.4     | T0 (Timer 0 external input)            |
| P3.5     | T1 (Timer 1 external input)            |
| P3.6     | WR (external data memory write strobe) |
| P3.7     | RD (external data memory read strobe)  |

The Port 3 output buffers can source/sink 4 LS TTL loads.

**RST:** Reset input. A high on this pin for two machine cycles while the oscillator is running resets the device.

**ALE/PROG:** Address Latch Enable output pulse for latching the low byte of the address during accesses to external memory. ALE is emitted at a constant rate of 1/6 of the oscillator frequency, for external timing or clocking purposes, even when there are no accesses to external memory. (However, one ALE pulse is skipped during each access to external Data Memory). This pin is also the program pulse input (PROG) during EPROM programming.

**PSEN:** Program Store Enable is the read strobe to external Program Memory. When the device is executing out of external Program Memory, PSEN is activated twice each machine cycle (except that two PSEN activations are skipped during accesses to external Data Memory). PSEN is not activated when the device is executing out of internal Program Memory.

**EA/VPP:** When EA is held high the CPU executes out of internal Program Memory (unless the Program Counter exceeds 0FFFFH in the 8051AH, or 1FFFFH in the 8052). Holding EA low forces the CPU to execute out of external memory regardless of the Program Counter value. In the 8031AH and 8032, EA must be externally wired low. In the 8751H, this pin also receives the 21V programming supply voltage (VPP) during EPROM programming.

**XTAL1:** Input to the inverting oscillator amplifier.

**XTAL2:** Output from the inverting oscillator amplifier.

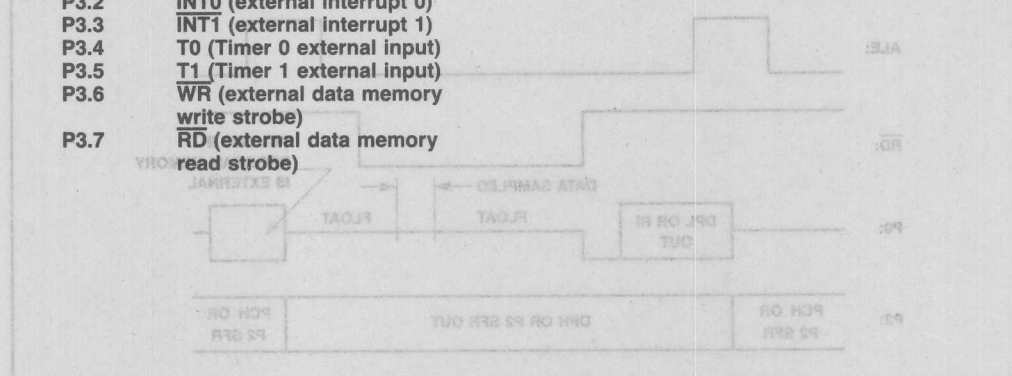


Figure 7-38. External Data Memory Read Cycle





# MCS®-51 ARCHITECTURE

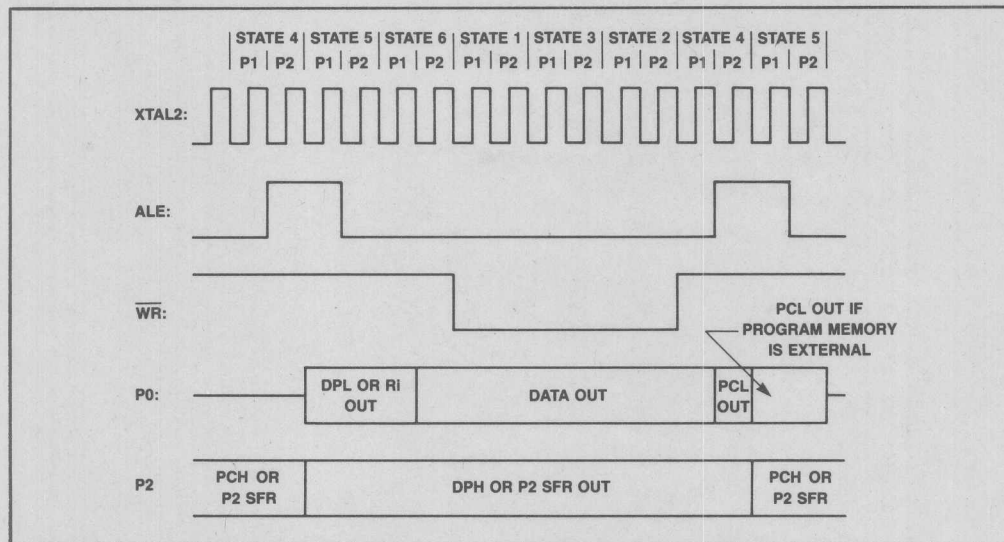


Figure 7-40. External Data Memory Write Cycle

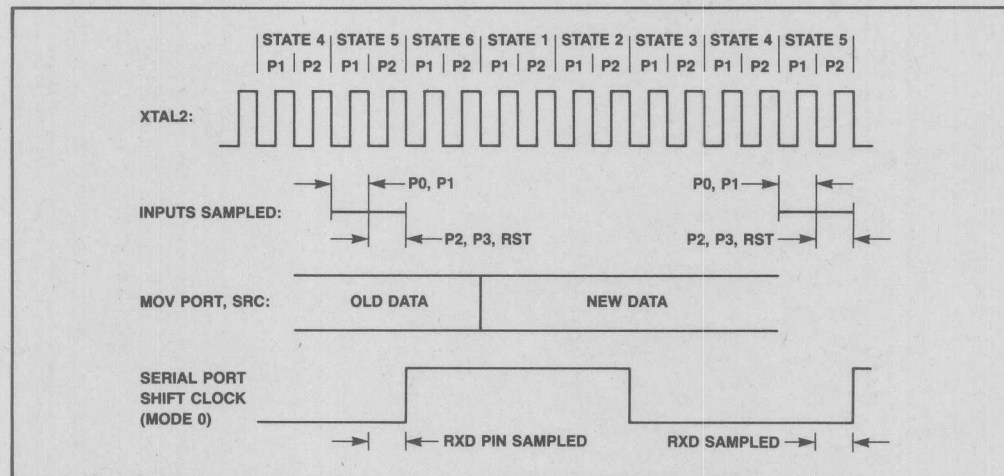


Figure 7-41. Port Operation



# and Instruction Set

---







# CHAPTER 8

## MCS®-51 PROGRAMMER'S GUIDE AND INSTRUCTION SET

The information presented in this chapter is collected from chapter 7 (MCS®-51 ARCHITECTURE) of this book. The material has been selected and rearranged to form a quick and convenient reference for the programmers of the MCS-51.

The following list should make it easier to find a subject in this chapter.

### MEMORY ORGANIZATION

|  |      |
|--|------|
| PROGRAM MEMORY .....                             | 8.2  |
| DATA MEMORY .....                                | 8.3  |
| DIRECT AND INDIRECT ADDRESS AREA .....           | 8.5  |
| SPECIAL FUNCTION REGISTERS .....                 | 8.7  |
| CONTENTS OF SFRs AFTER POWER-ON .....            | 8.8  |
| SFR MEMORY MAP .....                             | 8.9  |
| PROGRAM STATUS WORD (PSW) .....                  | 8.10 |
| POWER CONTROL REGISTER (PCON) .....              | 8.11 |
| INTERRUPTS .....                                 | 8.12 |
| INTERRUPT ENABLE REGISTER (IE) .....             | 8.13 |
| ASSIGNING PRIORITY LEVEL .....                   | 8.14 |
| INTERRUPT PRIORITY REGISTER .....                | 8.14 |
| TIMER/COUNTER CONTROL REGISTER (TCON) .....      | 8.15 |
| TIMER/COUNTER MODE CONTROL REGISTER (TMOD) ..... | 8.16 |
| TIMER SET-UP .....                               | 8.17 |
| TIMER/COUNTER 0 .....                            | 8.18 |
| TIMER/COUNTER 1 .....                            | 8.19 |
| TIMER/COUNTER 2 CONTROL REGISTER (T2CON) .....   | 8.20 |
| TIMER/COUNTER 2 SET-UP .....                     | 8.21 |
| SERIAL PORT CONTROL REGISTER .....               | 8.22 |
| SERIAL PORT SET-UP .....                         | 8.23 |
| GENERATING BAUD RATES .....                      | 8.23 |
| MCS-51 INSTRUCTION SET .....                     | 8.26 |
| INSTRUCTION DEFINITIONS .....                    | 8.33 |

## MEMORY ORGANIZATION

## PROGRAM MEMORY:

The 8051 has separate address spaces for Program Memory and Data Memory. The Program Memory can be up to 64K bytes long. The lower 4K (8K for the 8052) may reside on-chip.

Figure 8.1 shows a map of the 8051 program memory, and Figure 8.2 shows a map of the 8052 program memory.

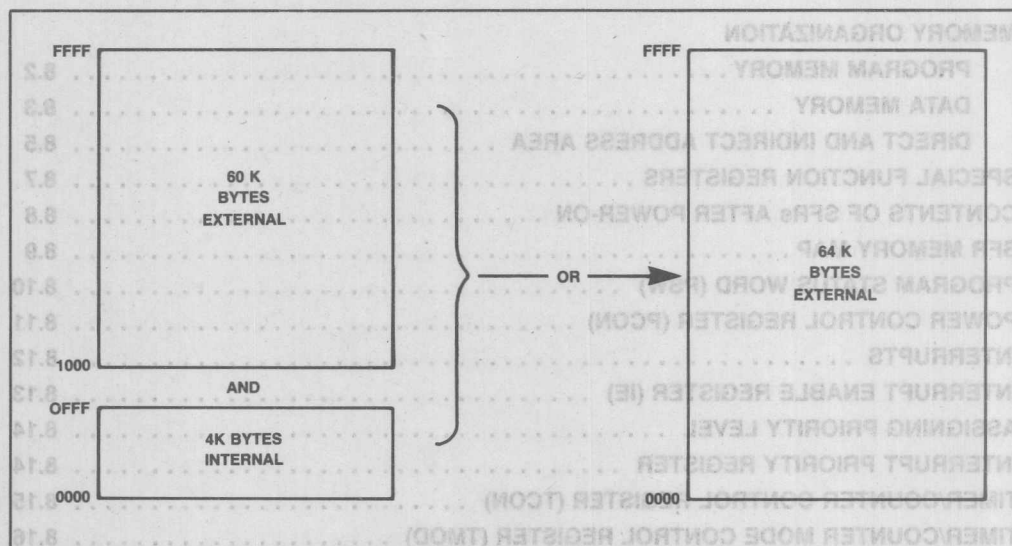


Figure 8.1  
The 8051 Program Memory.

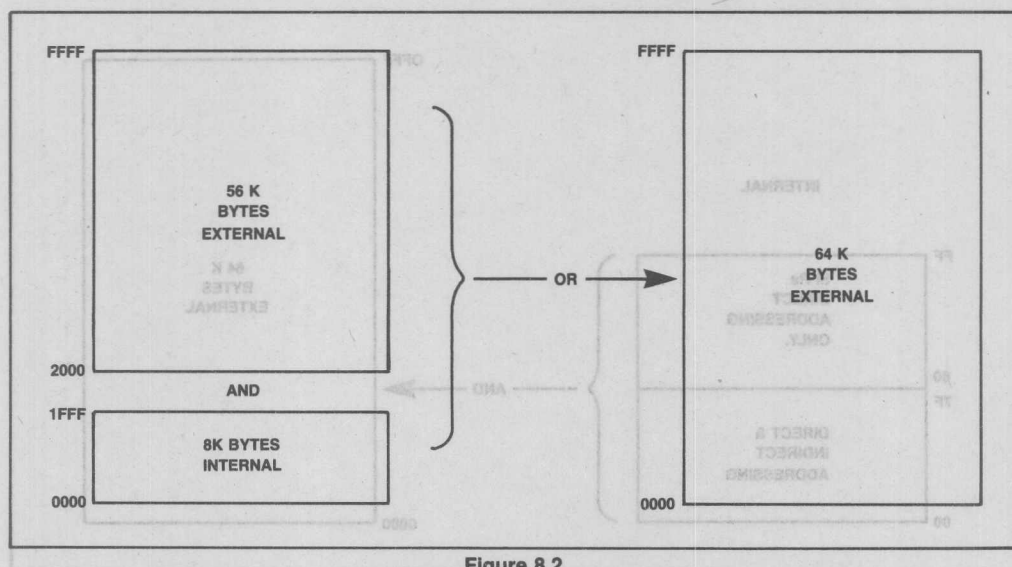


Figure 8.2  
The 8052 Program Memory.

### DATA MEMORY:

The 8051 can address up to 64K bytes of Data Memory external to the chip. The "MOVX" instruction is used to access the external data memory. (Refer to the MCS-51 Instruction Set, in this chapter, for detailed description of instructions).

The 8051 has 128 bytes of on-chip RAM (256 bytes in the 8052) plus a number of Special Function Registers (SFRs). The lower 128 bytes of RAM can be accessed either by direct addressing (MOV data addr) or by indirect addressing (MOV @Ri). Figure 8.3 shows the 8051 and the 8052 Data Memory organization.

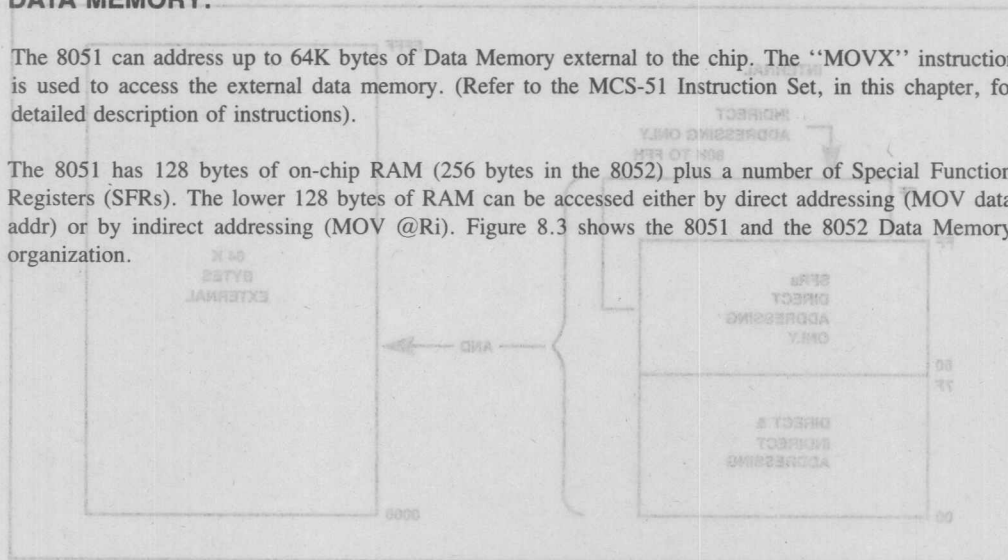


Figure 8.3  
The 8052 Data Memory.

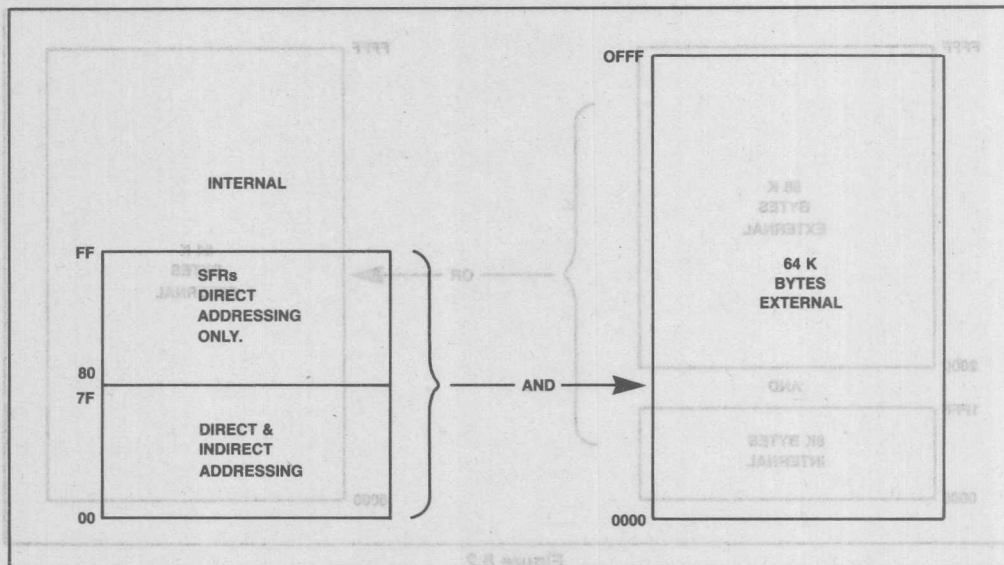


Figure 8.3a  
The 8051 Data Memory.

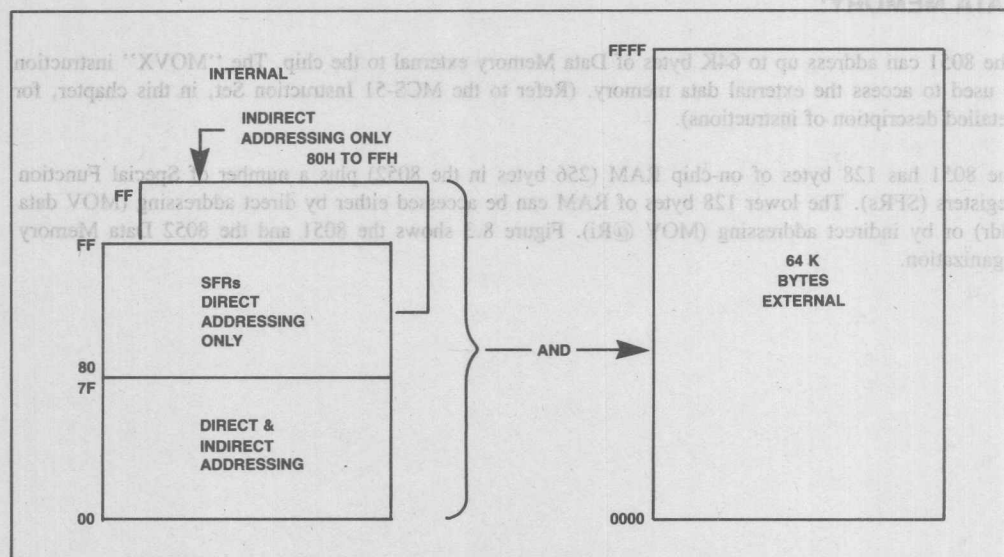


Figure 8.3b  
The 8052 Data Memory.



**INDIRECT ADDRESS AREA:**

Note that in Figure 8.3b the SFRs and the indirect address RAM have the same addresses (80H-0FFH). Nevertheless, they are two separate areas and are accessed in two different ways.

For example the instruction

```
MOV 80H,#0AAH
```

writes 0AAH to Port 0 which is one of the SFRs and the instruction

```
MOV R0,#80H
```

```
MOV @R0,#0BBH
```

writes 0BBH in location 80H of the data RAM. Thus, after execution of both of the above instructions Port 0 will contain 0AAH and location 80 of the RAM will contain 0BBH.

**DIRECT AND INDIRECT ADDRESS AREA:**

The 128 bytes of RAM which can be accessed by both direct and indirect addressing can be divided into 3 segments as listed below and shown in Figure 8.4.

**1. Register Banks 0-3:** Locations 0 through 1FH (32 bytes). ASM-51 and the device after reset default to register bank 0. To use the other register banks the user must select them in the software (refer to the MCS-51 Micro Assembler User's Guide). Each register bank contains 8 one-byte registers, 0 through 7.

Reset initializes the Stack Pointer to location 07H and it is incremented once to start from location 08H which is the first register (R0) of the second register bank. Thus, if one is going to use more than one register bank, the SP should be initialized to a different location of the RAM where is not used for data storage (ie, higher part of the RAM).

**2. Bit Addressable Area:** 16 bytes have been assigned for this segment, 20H-2FH. Each one of the 128 bits of this segment can be directly addressed (0-7FH).

The bits can be referred to in two ways both of which are acceptable by the ASM-51. One way is to refer to their addresses, ie, 0 to 7FH. The other way is with reference to bytes 20H to 2FH. Thus bits 0-7 can also be referred to as bits 20.0-20.7, and bits 8-FH are the same as 21.0-21.7 and so on.

Each of the 16 bytes in this segment can also be addressed as a byte.

**3. Scratch Pad Area:** Bytes 30H through 7FH are available to the user as data RAM. However, if the stack pointer has been initialized to this area, enough number of bytes should be left aside for it to prevent SP data destruction.

Figure 8.4 shows the different segments of the on-chip RAM.

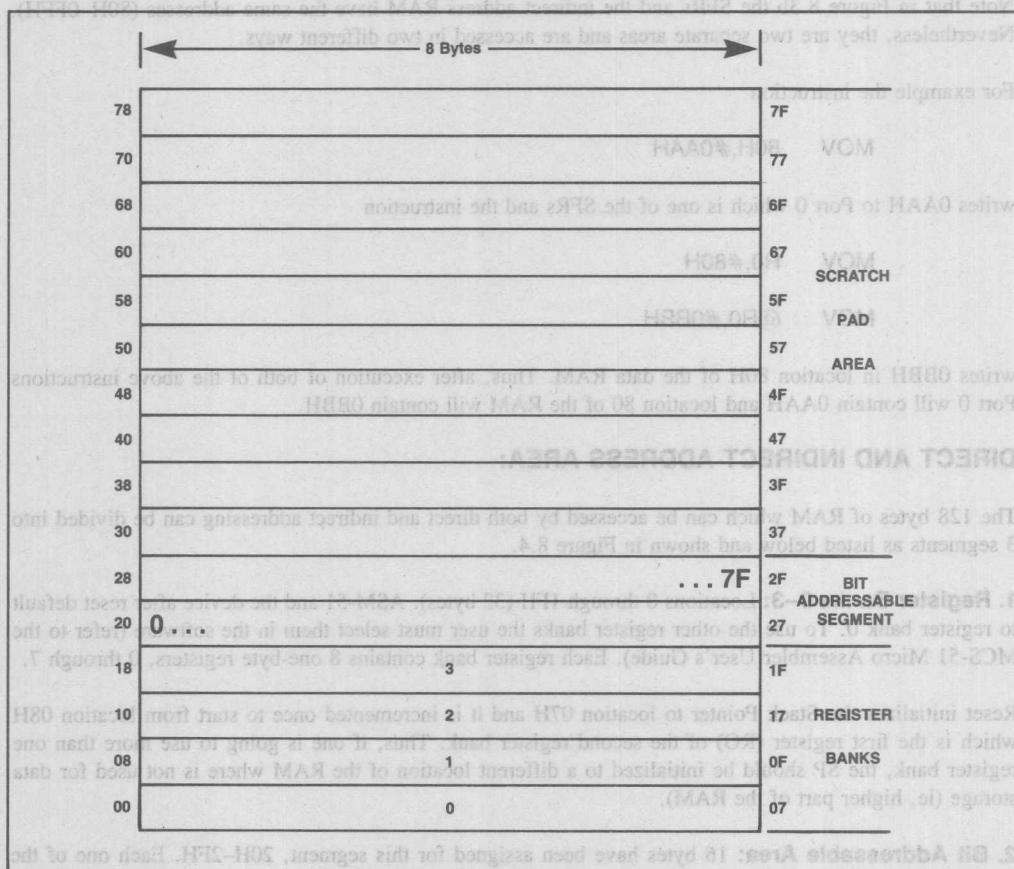


Figure 8.4

#### 128 Bytes of RAM Direct And Indirect Addressable

The bits can be referred to in two ways both of which are acceptable by the ASM-51. One way is to refer to their addresses, i.e. 0 to 7FH. The other way is with reference to bytes 30H to 7FH. Thus bits 0-7 can also be referred to as bits 20-20.7, and bits 8-7FH are the same as 21-21.7 and so on.

Each of the 16 bytes in this segment can also be addressed as a byte.

**3. Scratch Pad Area:** Bytes 30H through 7FH are available to the user as data RAM. However, if the stack pointer has been initialized to this area, enough number of bytes should be left aside for it to prevent SP data destruction.

**SPECIAL FUNCTION REGISTERS:**

Table 8.1 contains a list of all the SFRs, and their addresses. The table also indicates whether each one is only byte addressable (doesn't have a designator), or byte and bit addressable (marked with an asterisk). Furthermore, it designates those available only in the 8052 (marked with a '+' sign).

Comparing Table 8.1 and Figure 8.5 shows that all of the SFRs that are byte and bit addressable are located on the first column of the diagram in Figure 8.5.

| Table 8.1 |                              |         |
|-----------|------------------------------|---------|
| SYMBOL    | NAME                         | ADDRESS |
| *ACC      | Accumulator                  | 0E0H    |
| *B        | B Register                   | 0F0H    |
| *PSW      | Program Status Word          | 0D0H    |
| SP        | Stack Pointer                | 81H     |
| DPTR      | Data Pointer 2 Bytes:        |         |
| DPL       | Low Byte                     | 82H     |
| DPH       | High Byte                    | 83H     |
| *P0       | Port 0                       | 80H     |
| *P1       | Port 1                       | 90H     |
| *P2       | Port 2                       | 0A0H    |
| *P3       | Port 3                       | 0B0H    |
| *IP       | Interrupt Priority Control   | 0B8H    |
| *IE       | Interrupt Enable Control     | 0A8H    |
| TMOD      | Timer/Counter Mode Control   | 89H     |
| *TCON     | Timer/Counter Control        | 88H     |
| *+T2CON   | Timer/Counter 2 Control      | 0C8H    |
| TH0       | Timer/Counter 0 High Byte    | 8CH     |
| TL0       | Timer/Counter 0 Low Byte     | 8AH     |
| TH1       | Timer/Counter 1 High Byte    | 8DH     |
| TL1       | Timer/Counter 1 Low Byte     | 8BH     |
| +TH2      | Timer/Counter 2 High Byte    | 0CDH    |
| +TL2      | Timer/Counter 2 Low Byte     | 0CCH    |
| +RCAP2H   | T/C 2 Capture Reg. High Byte | 0CBH    |
| +RCAP2L   | T/C 2 Capture Reg. Low Byte  | 0CAH    |
| *SCON     | Serial Control               | 98H     |
| SBUF      | Serial Data Buffer           | 99H     |
| PCON      | Power Control                | 87H     |

## WHAT DO THE SFRs CONTAIN JUST AFTER POWER-ON OR A RESET?

Table 8.2 lists the contents of each SFR after power-on or a hardware reset.

**Table 8.2**

**Contents of the SFRs after reset.**

| REGISTER | VALUE IN BINARY                 |
|----------|---------------------------------|
| *ACC     | 00000000                        |
| *B       | 00000000                        |
| *PSW     | 00000000                        |
| SP       | 00000111                        |
| DPTR     |                                 |
| DPH      | 00000000                        |
| DPL      | 00000000                        |
| *P0      | 11111111                        |
| *P1      | 11111111                        |
| *P2      | 11111111                        |
| *P3      | 11111111                        |
| *IP      | 8051 XXX00000,<br>8052 XX000000 |
| *IE      | 8051 0XX00000,<br>8052 0X000000 |
| TMOD     | 00000000                        |
| *TCON    | 00000000                        |
| *+T2CON  | 00000000                        |
| TH0      | 00000000                        |
| TL0      | 00000000                        |
| TH1      | 00000000                        |
| TL1      | 00000000                        |
| +TH2     | 00000000                        |
| +TL2     | 00000000                        |
| +RCAP2H  | 00000000                        |
| +RCAP2L  | 00000000                        |
| *SCON    | 00000000                        |
| SBUF     | Indeterminate                   |
| PCON     | HMOS 0XXXXXXX<br>CHMOS 0XXX0000 |

X = Undefined  
 \* = Bit Addressable  
 += 8052 only



## SFR MEMORY MAP

8 Bytes

|    |       |      |        |        |     |     |  |  |      |    |
|----|-------|------|--------|--------|-----|-----|--|--|------|----|
| F8 |       |      |        |        |     |     |  |  |      | FF |
| F0 | B     |      |        |        |     |     |  |  |      | F7 |
| E8 |       |      |        |        |     |     |  |  |      | EF |
| E0 | ACC   |      |        |        |     |     |  |  |      | E7 |
| D8 |       |      |        |        |     |     |  |  |      | DF |
| D0 | PSW   |      |        |        |     |     |  |  |      | D7 |
| C8 | T2CON |      | RCAP2L | RCAP2H | TL2 | TH2 |  |  |      | CF |
| C0 |       |      |        |        |     |     |  |  |      | C7 |
| B8 | IP    |      |        |        |     |     |  |  |      | BF |
| B0 | P3    |      |        |        |     |     |  |  |      | B7 |
| A8 | IE    |      |        |        |     |     |  |  |      | AF |
| A0 | P2    |      |        |        |     |     |  |  |      | A7 |
| 98 | SCON  | SBUF |        |        |     |     |  |  |      | 9F |
| 90 | P1    |      |        |        |     |     |  |  |      | 97 |
| 88 | TCON  | TMOD | TL0    | TL1    | TH0 | TH1 |  |  |      | 8F |
| 80 | P0    | SP   | DPL    | DPH    |     |     |  |  | PCON | 87 |

Figure 8.5

| ADDRESS | REGISTER BANK | R50 | R51 |
|---------|---------------|-----|-----|
| 00H-07H | 0             | 0   | 0   |
| 08H-0FH | 1             | 1   | 0   |
| 10H-17H | 2             | 0   | 1   |
| 18H-1FH | 3             | 1   | 1   |

Those SFRs that have their bits assigned for various functions are listed in this section. A brief description of each bit is provided for quick reference. For more detailed information refer to the Architecture Chapter of this book.

**PSW: PROGRAM STATUS WORD. BIT ADDRESSABLE.**

|     | CY     | AC   | F0 | RS1 | RS0 | OV | — | P |
|-----|--------|--|----|-----|-----|----|---|---|
| CY  | PSW. 7 | Carry Flag.  |    |     |     |    |   |   |
| AC  | PSW. 6 | Auxiliary Carry Flag.  |    |     |     |    |   |   |
| F0  | PSW. 5 | Flag 0 available to the user for general purpose.  |    |     |     |    |   |   |
| RS1 | PSW. 4 | Register Bank selector bit 1 (SEE NOTE 1).   |    |     |     |    |   |   |
| RS0 | PSW. 3 | Register Bank selector bit 0 (SEE NOTE 1).   |    |     |     |    |   |   |
| OV  | PSW. 2 | Overflow Flag.   |    |     |     |    |   |   |
| —   | PSW. 1 | Reserved for future use.   |    |     |     |    |   |   |
| P   | PSW. 0 | Parity flag. set/cleared by hardware each instruction cycle to indicate an odd/even number of '1' bits in the accumulator. |    |     |     |    |   |   |

**NOTE 1:** The value presented by RS0 and RS1 selects the corresponding register bank.

| RS1 | RS0 | REGISTER BANK | ADDRESS |
|-----|-----|---------------|---------|
| 0   | 0   | 0             | 00H–07H |
| 0   | 1   | 1             | 08H–0FH |
| 1   | 0   | 2             | 10H–17H |
| 1   | 1   | 3             | 18H–1FH |

**PCON: POWER CONTROL REGISTER. NOT BIT ADDRESSABLE.**

|      |   |   |   |     |     |    |     |
|------|---|---|---|-----|-----|----|-----|
| SMOD | — | — | — | GF1 | GF0 | PD | IDL |
|------|---|---|---|-----|-----|----|-----|

**SMOD** Double baud rate bit. When Timer 1 is used to generate baud rate and  $SMOD = 1$ , the baud rate is doubled when the Serial Port is used in modes 1, 2, or 3.

— Not implemented.

— Not implemented.

— Not implemented.

**GF1** General purpose flag bit.

**GF0** General purpose flag bit.

**PD** Power Down bit. Setting this bit activates Power Down operation in the 80C51BH. (Available only in CHMOS).

**IDL** Idle Mode bit. Setting this bit activates Idle Mode operation in the 80C51BH. (Available only in CHMOS).

If 1s are written to PD and IDL at the same time, PD takes precedence.

## INTERRUPTS:

In order to use any of the interrupts in the MCS-51, one must take all three of the following steps.

1. Set the EA (enable all) bit in the IE register to 1.
2. Set the corresponding individual interrupt enable bit in the IE register to 1.
3. Begin the interrupt service routine at the corresponding Vector Address of that interrupt. See Table below.

| INTERRUPT<br>SOURCE | VECTOR<br>ADDRESS |
|---------------------|-------------------|
| IE0                 | 0003H             |
| TF0                 | 000BH             |
| IE1                 | 0013H             |
| TF1                 | 001BH             |
| RI & TI             | 0023H             |
| TF2 & EXF2          | 002BH             |

In addition, for external interrupts, pins  $\overline{\text{INT0}}$  and  $\overline{\text{INT1}}$  (P3.2 and P3.3) must be set to 1, and depending on whether the interrupt is to be level or transition activated, bits IT0 or IT1 in the TCON register may need to be set to 1.

ITx = 0 level activated  
ITx = 1 transition activated



# IE: INTERRUPT ENABLE REGISTER. BIT ADDRESSABLE

If the bit is 0, the corresponding interrupt is disabled. If the bit is 1, the corresponding interrupt is enabled.

|    |   |     |    |     |     |     |     |
|----|---|-----|----|-----|-----|-----|-----|
| EA | — | ET2 | ES | ET1 | EX1 | ET0 | EX0 |
|----|---|-----|----|-----|-----|-----|-----|

- EA IE. 7 Disables all interrupts. If EA = 0, no interrupt will be acknowledged. If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.
- IE. 6 Not implemented.
- ET2 IE. 5 Enable or disable the Timer 2 overflow or capture interrupt (8052 only).
- ES IE. 4 Enable or disable the serial port interrupt.
- ET1 IE. 3 Enable or disable the Timer 1 overflow interrupt.
- EX1 IE. 2 Enable or disable External Interrupt 1.
- ET0 IE. 1 Enable or disable the Timer 0 overflow interrupt.
- EX0 IE. 0 Enable or disable External Interrupt 0.

## IP: INTERRUPT PRIORITY REGISTER. BIT ADDRESSABLE

If the bit is 0, the corresponding interrupt has a lower priority and if the bit is 1 the corresponding interrupt has a higher priority.

|   |   |     |     |     |     |     |     |
|---|---|-----|-----|-----|-----|-----|-----|
| — | — | PT2 | PT1 | PT0 | PT2 | PT1 | PT0 |
|---|---|-----|-----|-----|-----|-----|-----|

- IP. 7 Not implemented.
- IP. 6 Not implemented.
- PT2 IP. 5 Defines the Timer 2 interrupt priority level (8052 only).
- PT1 IP. 4 Defines the Timer 1 interrupt priority level.
- PT0 IP. 3 Defines the Timer 0 interrupt priority level.
- PT2 IP. 2 Defines the External Interrupt 2 priority level.
- PT1 IP. 1 Defines the External Interrupt 1 priority level.
- PT0 IP. 0 Defines the External Interrupt 0 priority level.

**ASSIGNING HIGHER PRIORITY TO ONE OR MORE INTERRUPTS:**

In order to assign higher priority to an interrupt the corresponding bit in the IP register must be set to 1.

Remember that while an interrupt service is in progress, it cannot be interrupted by a lower or same level interrupt.

|    |   |     |    |     |     |     |     |
|----|---|-----|----|-----|-----|-----|-----|
| EA | — | ET2 | ES | ET1 | EX1 | ET0 | EX0 |
|----|---|-----|----|-----|-----|-----|-----|

**PRIORITY WITHIN LEVEL:**

Priority within level is only to resolve simultaneous requests of the same priority level.

From high to low, interrupt sources are listed below:

IE0  
TF0  
IE1  
TF1  
RI or TI  
TF2 or EXF2

**IP: INTERRUPT PRIORITY REGISTER. BIT ADDRESSABLE.**

If the bit is 0, the corresponding interrupt has a lower priority and if the bit is 1 the corresponding interrupt has a higher priority.

|   |   |     |    |     |     |     |     |
|---|---|-----|----|-----|-----|-----|-----|
| — | — | PT2 | PS | PT1 | PX1 | PT0 | PX0 |
|---|---|-----|----|-----|-----|-----|-----|

- IP. 7 Not implemented.
- IP. 6 Not implemented.
- PT2 IP. 5 Defines the Timer 2 interrupt priority level (8052 only).
- PS IP. 4 Defines the Serial Port interrupt priority level.
- PT1 IP. 3 Defines the Timer 1 interrupt priority level.
- PX1 IP. 2 Defines the External Interrupt 1 priority level.
- PT0 IP. 1 Defines the Timer 0 interrupt priority level.
- PX0 IP. 0 Defines the External Interrupt 0 priority level.

**TCON: TIMER/COUNTER CONTROL REGISTER. BIT ADDRESSABLE.**

|     |     |     |     |     |     |     |     |
|-----|-----|-----|-----|-----|-----|-----|-----|
| TF1 | TR1 | TF0 | TR0 | IE1 | IT1 | IE0 | IT0 |
|-----|-----|-----|-----|-----|-----|-----|-----|

- TF1 TCON. 7 Timer 1 overflow flag. Set by hardware when the Timer/Counter 1 overflows. Cleared by hardware as processor vectors to the interrupt service routine.
- TR1 TCON. 6 Timer 1 run control bit. Set/cleared by software to turn Timer/Counter 1 ON/OFF.
- TF0 TCON. 5 Timer 0 overflow flag. Set by hardware when the Timer/Counter 0 overflows. Cleared by hardware as processor vectors to the service routine.
- TR0 TCON. 4 Timer 0 run control bit. Set/cleared by software to turn Timer/Counter 0 ON/OFF.
- IE1 TCON. 3 External Interrupt 1 edge flag. Set by hardware when External Interrupt edge is detected, cleared by hardware when interrupt is processed.
- IT1 TCON. 2 Interrupt 1 type control bit. Set/cleared by software to specify falling edge/low level triggered External Interrupt.
- IE0 TCON. 1 External Interrupt 0 edge flag. Set by hardware when External Interrupt edge detected, cleared by hardware when interrupt is processed.
- IT0 TCON. 0 Interrupt 0 type control bit. Set/cleared by software to specify falling edge/low level triggered External Interrupt.

**TMOD: TIMER/COUNTER MODE CONTROL REGISTER. NOT BIT ADDRESSABLE.**

|      |     |    |    |      |     |    |    |
|------|-----|----|----|------|-----|----|----|
| GATE | C/T | M1 | M0 | GATE | C/T | M1 | M0 |
|------|-----|----|----|------|-----|----|----|

**TIMER 1**

**TIMER 0**

**GATE** When TRx (in TCON) is set and GATE = 1, TIMER/COUNTERx will run only while INTx pin is high (hardware control). When GATE = 0, TIMER/COUNTERx will run only while TRx = 1 (software control).

**C/T** Timer or counter selector. Cleared for Timer operation (input from internal system clock). Set for Counter operation (input from Tx input pin).

**M1** Mode selector bit. (NOTE 1)

**M0** Mode selector bit. (NOTE 1)

**NOTE 1:**

**M1 M0 OPERATING MODE**

0 0 0 13-bit Timer (MCS-48 compatible)

0 1 1 16-bit Timer/Counter

1 0 2 8-bit Auto-Reload Timer/Counter

1 1 3 (Timer 0) TL0 is an 8-bit Timer/Counter controlled by the standard Timer 0 control bits, TH0 is an 8-bit Timer and is controlled by Timer 1 control bits.

1 1 3 (Timer 1) Timer/Counter 1 stopped.



TIMER SET-UP

Tables 8.3 through 8.6 give some values for TMOD which can be used to set up Timer 0 in different modes.

It is assumed that only one timer is being used at a time. If it is desired to run Timers 0 and 1 simultaneously, in any mode, the value in TMOD for Timer 0 must be ORed with the value shown for Timer 1 (tables 8.5 and 8.6).

For example if it is desired to run Timer 0 in mode 1 GATE (external control), and Timer 1 in mode 2 COUNTER, then the value that must be loaded into TMOD is 69H (09H from Table 8.3 ORed with 60H from Table 8.6).

Moreover it is assumed that the user, at this point, is not ready to turn the timers on and will do that at a different point in the program by setting bit TRx (in TCON) to 1.

| MODE | FUNCTION          | (NOTE 1) | (NOTE 2) |
|------|-------------------|----------|----------|
| 0    | 13-bit timer      | 04H      | 0CH      |
| 1    | 16-bit timer      | 05H      | 0DH      |
| 2    | 8-bit auto reload | 06H      | 0EH      |
| 3    | one 8-bit counter | 07H      | 0FH      |

| MODE | COUNTER 0 FUNCTION | INTERNAL CONTROL (NOTE 1) | EXTERNAL CONTROL (NOTE 2) |
|------|--------------------|---------------------------|---------------------------|
| 0    | 13-bit timer       | 04H                       | 0CH                       |
| 1    | 16-bit timer       | 05H                       | 0DH                       |
| 2    | 8-bit auto reload  | 06H                       | 0EH                       |
| 3    | one 8-bit counter  | 07H                       | 0FH                       |

NOTE 1: The Timer is turned ON/OFF by setting/clearing bit TR0 in the software.  
NOTE 2: The Timer is turned ON/OFF by the 1 to 0 transition on INT0 (P1.2) when TR0 = 1, (hardware control).

## TIMER/COUNTER 0

## AS A TIMER:

Table 8.3

| MODE | TIMER 0<br>FUNCTION | TMOD                            |                                 |
|------|---------------------|---------------------------------|---------------------------------|
|      |                     | INTERNAL<br>CONTROL<br>(NOTE 1) | EXTERNAL<br>CONTROL<br>(NOTE 2) |
| 0    | 13-bit timer        | 00H                             | 08H                             |
| 1    | 16-bit timer        | 01H                             | 09H                             |
| 2    | 8-bit auto reload   | 02H                             | 0AH                             |
| 3    | two 8-bit timers    | 03H                             | 0BH                             |

## AS A COUNTER:

Table 8.4

| MODE | COUNTER 0<br>FUNCTION | TMOD                            |                                 |
|------|-----------------------|---------------------------------|---------------------------------|
|      |                       | INTERNAL<br>CONTROL<br>(NOTE 1) | EXTERNAL<br>CONTROL<br>(NOTE 2) |
| 0    | 13-bit timer          | 04H                             | 0CH                             |
| 1    | 16-bit timer          | 05H                             | 0DH                             |
| 2    | 8-bit auto reload     | 06H                             | 0EH                             |
| 3    | one 8-bit counter     | 07H                             | 0FH                             |

**NOTE 1:** The Timer is turned ON/OFF by setting/clearing bit TR0 in the software.

**NOTE 2:** The Timer is turned ON/OFF by the 1 to 0 transition on  $\overline{\text{INT0}}$  (P3.2) when TR0 = 1, (hardware control).

## TIMER/COUNTER 1

## AS A TIMER:

Table 8.5

| MODE | TIMER 1<br>FUNCTION | TMOD                            |                                 |
|------|---------------------|---------------------------------|---------------------------------|
|      |                     | INTERNAL<br>CONTROL<br>(NOTE 1) | EXTERNAL<br>CONTROL<br>(NOTE 2) |
| 0    | 13-bit timer        | 00H                             | 80H                             |
| 1    | 16-bit timer        | 10H                             | 90H                             |
| 2    | 8-bit auto reload   | 20H                             | A0H                             |
| 3    | does not run        | 30H                             | B0H                             |

## AS A COUNTER:

Table 8.6

| MODE | COUNTER 1<br>FUNCTION | TMOD                            |                                 |
|------|-----------------------|---------------------------------|---------------------------------|
|      |                       | INTERNAL<br>CONTROL<br>(NOTE 1) | EXTERNAL<br>CONTROL<br>(NOTE 1) |
| 0    | 13-bit timer          | 40H                             | C0H                             |
| 1    | 16-bit timer          | 50H                             | D0H                             |
| 2    | 8-bit auto reload     | 60H                             | E0H                             |
| 3    | not available         | —                               | —                               |

**NOTE 1:** The Timer is turned ON/OFF by setting/clearing bit TR1 in the software.

**NOTE 2:** The Timer is turned ON/OFF by the 1 to 0 transition on INT1 (P3.3) when TR1 = 1, (hardware control).

# **T2CON: TIMER/COUNTER 2 CONTROL REGISTER. BIT ADDRESSABLE,**

**8052 ONLY.**

| TF2 | EXF2 | RCLK | TCLK | EXEN2 | TR2 | C/ $\overline{T}2$ | CP/ $\overline{RL}2$ |
|-----|------|------|------|-------|-----|--------------------|----------------------|
|-----|------|------|------|-------|-----|--------------------|----------------------|

|                      |         |  |  |  |  |  |  |
|----------------------|---------|--|--|--|--|--|--|
| TF2                  | T2CON.7 | Timer 2 overflow flag set by hardware and cleared by software. TF2 will not set when either RCLK = 1 or CLK = 1  |  |  |  |  |  |
| EXF2                 | T2CON.6 | Timer 2 external flag set when either a capture or reload is caused by a negative transition on T2EX, and EXEN2 = 1. When Timer 2 interrupt is enabled, EXF2 = 1 will cause the cpu to vector to the Timer 2 interrupt routine. EXF2 must be cleared by software.  |  |  |  |  |  |
| RCLK                 | T2CON.5 | Receive clock flag. When set, causes the Serial Port to use Timer 2 overflow pulses for its receive clock in modes 1 & 3. RCLK = 0 causes Timer 1 overflow to be used for the receive clock.   |  |  |  |  |  |
| TCLK                 | T2CON.4 | Transmit clock flag. When set, causes the Serial Port to use Timer 2 overflow pulses for its transmit clock in modes 1 & 3. TCLK = 0 causes Timer 1 overflows to be used for the transmit clock.   |  |  |  |  |  |
| EXEN2                | T2CON.3 | Timer 2 external enable flag. When set, allows a capture or reload to occur as a result of negative transition on T2EX if Timer 2 is not being used to clock the Serial Port. EXEN2 = 0 causes Timer 2 to ignore events at T2EX.   |  |  |  |  |  |
| TR2                  | T2CON.2 | Software START/STOP control for Timer 2. A logic 1 starts the Timer.   |  |  |  |  |  |
| C/ $\overline{T}2$   | T2CON.1 | Timer or Counter select.<br>0 = Internal Timer. 1 = External Event Counter (falling edge triggered).   |  |  |  |  |  |
| CP/ $\overline{RL}2$ | T2CON.0 | Capture/Reload flag. When set, captures will occur on negative transitions at T2EX if EXEN2 = 1. When cleared, Auto-Reloads will occur either with Timer 2 overflows or negative transitions at T2EX when EXEN2 = 1. When either RCLK = 1 or TCLK = 1, this bit is ignored and the Timer is forced to Auto-Reload on Timer 2 overflow. |  |  |  |  |  |



**TIMER/COUNTER 2 SET-UP**

Except for the baud rate generator mode, the values given for T2CON do not include the setting of the TR2 bit. Therefore, bit TR2 must be set, separately, to turn the timer on.

**AS A TIMER:****Table 8.7**

| MODE  | T2CON                     |                           |
|---|---------------------------|---------------------------|
|   | INTERNAL CONTROL (NOTE 1) | EXTERNAL CONTROL (NOTE 2) |
| 16-bit auto-reload                                    | 00H                       | 08H                       |
| 16-bit capture  | 01H                       | 09H                       |
| BAUD rate generator receive & transmit same baud rate | 34H                       | 36H                       |
| receive only  | 24H                       | 26H                       |
| transmit only   | 14H                       | 16H                       |

**AS A COUNTER:****Table 8.8**

| MODE               | T2CON                     |                           |
|--------------------|---------------------------|---------------------------|
|                    | INTERNAL CONTROL (NOTE 1) | EXTERNAL CONTROL (NOTE 2) |
| 16-bit auto-reload | 02H                       | 0AH                       |
| 16-bit capture     | 03H                       | 0BH                       |

**NOTE 1:** Capture/Reload occurs only on Timer/Counter overflow.

**NOTE 2:** Capture/Reload occurs on Timer/Counter overflow and a 1 to 0 transition on T2EX (P1.1) pin except when Timer 2 is used in the baud rate generating mode.

**SCON: SERIAL PORT CONTROL REGISTER. BIT ADDRESSABLE.**

|     |     |     |     |     |     |    |    |
|-----|-----|-----|-----|-----|-----|----|----|
| SM0 | SM1 | SM2 | REN | TB8 | RB8 | TI | RI |
|-----|-----|-----|-----|-----|-----|----|----|

|     |        |  |
|-----|--------|--|
| SM0 | SCON.7 | Serial Port mode specifier. (NOTE 1).  |
| SM1 | SCON.6 | Serial Port mode specifier. (NOTE 1)   |
| SM2 | SCON.5 | Enables the multiprocessor communication feature in modes 2 & 3. In mode 2 or 3, if SM2 is set to 1 then RI will not be activated if the received 9th data bit (RB8) is 0. In mode 1, if SM2=1 then RI will not be activated if a valid stop bit was not received. In mode 0, SM2 should be 0. |
| REN | SCON.4 | Set/Cleared by software to Enable/Disable reception.   |
| TB8 | SCON.3 | The 9th bit that will be transmitted in modes 2 & 3. Set/Cleared by software.  |
| RB8 | SCON.2 | In modes 2 & 3 if the 9th data bit that was received. In mode 1, if SM2=0, RB8 is the stop bit that was received. In mode 0, TB8 is not used.  |
| TI  | SCON.1 | Transmit interrupt flag. Set by hardware at the end of the 8th bit time in mode 0, or at the beginning of the stop bit in the other modes. Must be cleared by software.  |
| RI  | SCON.0 | Receive interrupt flag. Set by hardware at the end of the 8th bit time in mode 0, or halfway through the stop bit time in the other modes (except see SM2). Must be cleared by software.   |

**NOTE 1:**

| SM0 | SM1 | MODE | DESCRIPTION    | BAUD RATE            |
|-----|-----|------|----------------|----------------------|
| 0   | 0   | 0    | SHIFT REGISTER | Fosc./12             |
| 0   | 1   | 1    | 8-Bit UART     | Variable             |
| 1   | 0   | 2    | 9-Bit UART     | Fosc./64 OR Fosc./32 |
| 1   | 1   | 3    | 9-Bit UART     | Variable             |

## SERIAL PORT SET-UP:

Table 8.9

| MODE | SCON | SM2 VARIATION                                |
|------|------|--|
| 0    | 10H  | SINGLE PROCESSOR<br>ENVIRONMENT<br>(SM2 = 0) |
| 1    | 50H  |  |
| 2    | 90H  |  |
| 3    | D0H  |  |
| 0    | NA   | MULTIPROCESSOR<br>ENVIRONMENT<br>(SM2 = 1)   |
| 1    | 70H  |  |
| 2    | B0H  |  |
| 3    | F0H  |  |

## GENERATING BAUD RATES

## SERIAL PORT IN MODE 0:

Mode 0 has a fixed baud rate which is 1/12 of the oscillator frequency. To run the serial port in this mode none of the timer/counters need to be set up. Only the SCON register needs to be defined.

$$\text{BAUD RATE} = \frac{\text{Osc Freq}}{12}$$

## SERIAL PORT IN MODE 1:

Mode 1 has a variable baud rate. The baud rate can be generated by either Timer 1 or Timer 2.

## USING TIMER/COUNTER 1 TO GENERATE BAUD RATES:

For this purpose timer 1 is used in mode 2 (auto reload). Refer to timer set up section of this chapter.

| MODE | BAUD RATE =   | K x Oscillator freq. |
|------|---|----------------------|
| 0    | $\frac{K \times \text{Oscillator freq.}}{32 \times 12 \times [256 - (\text{TH1})]}$ |                      |
| 1    |   |                      |
| 2    |   |                      |
| 3    |   |                      |
| 0    |   |                      |
| 1    |   |                      |
| 2    |   |                      |
| 3    |   |                      |

If SMOD = 0, then K = 1

If SMOD = 1, then K = 2. (SMOD is in the PCON register).

Most of the time the user knows the baud rate and needs to know the reload value for TH1. Therefore, the equation can be written as:

$$\text{TH1} = 256 - \frac{K \times \text{Osc freq.}}{384 \times \text{baud rate}}$$

Since the PCON register is not bit addressable, one way to set the bit is logical ORing the PCON register, (ie, ORL PCON, #80H). The address of PCON is 87H.

## USING TIMER/COUNTER 2 TO GENERATE BAUD RATES:

For this purpose timer 2 must be used in the baud rate generating mode (refer to timer 2 set up table in this chapter). If timer 2 is being clocked through pin T2 (P1.0) the baud rate is:

$$\text{Baud Rate} = \frac{\text{Timer 2 Overflow Rate}}{16}$$

And if it is being clocked internally the baud rate is:

$$\text{Baud Rate} = \frac{\text{Osc Freq}}{32 \times [65536 - (\text{RCAP2H}, \text{RCAP2L})]}$$



To obtain the reload value for RCAP2H and RCAP2L the above equation can be rewritten as:

$$\text{RCAP2H, RCAP2L} = 65536 - \frac{\text{Osc Freq}}{32 \times \text{Baud Rate}}$$

## SERIAL PORT IN MODE 2:

The baud rate is fixed in this mode and is 1/32 or 1/64 of the oscillator frequency depending on the value of the SMOD bit in the PCON register.

In this mode none of the timers are used and the clock comes from the internal phase 2 clock.

$$\text{SMOD} = 1, \text{Baud Rate} = 1/32 \text{ Osc Freq.}$$

$$\text{SMOD} = 0, \text{Baud Rate} = 1/64 \text{ Osc Freq.}$$

To set the SMOD bit: ORL PCON, #80H. The address of PCON is 87H.

## SERIAL PORT IN MODE 3:

The baud rate in mode 3 is variable and sets up exactly the same as in mode 1.

# MCS®-51 INSTRUCTION SET

## INTRODUCTION TO INSTRUCTION SET

The MCS®-51 instruction set includes 111 instructions, 49 of which are single-byte, 45 two-byte and 17 three byte. The instruction op code format consists of a function mnemonic followed by a "destination, source" operand field. This field specifies the data type and addressing method(s) to be used.

## FUNCTIONAL OVERVIEW

The MCS-51 instruction set is divided into four functional groups:

- Data Transfer
- Arithmetic
- Logic
- Control Transfer

### Data Transfer

Data transfer operations are divided into three classes;

- General Purpose
- Accumulator-Specific
- Address-Object

None of these operations affect the PSW flag settings except a POP or MOV directly to the PSW.

### GENERAL-PURPOSE TRANSFERS

- MOV performs a bit or a byte transfer from the source operand to the destination operand.
- PUSH increments the SP register and then transfers a byte from the source operand to the stack location currently addressed by SP.
- POP transfer a byte operand from the stack location addressed by SP to the destination operand and then decrements SP.

### ACCUMULATOR SPECIFIC TRANSFERS

- XCH exchanges the byte source operand with register A (accumulator).
- XCHD exchanges the low-order nibble of the byte source operand with the low-order nibble of A.

- MOVX performs a byte move between the External Data Memory and the accumulator. The external address can be specified by the DPTR register (16-bit) or the R1 or R0 register (8-bit).

- MOVC moves a byte from Program memory to the accumulator. The operand in A is used as an index into a 256-byte table pointed to by the base register (DPTR or PC). The byte operand accessed is transferred to the accumulator.

### ADDRESS-OBJECT TRANSFER

- MOV DPTR, #data loads 16-bits of immediate data into a pair of destination registers, DPH and DPL.

### Arithmetic

The 8051 has four basic mathematical operations. Only 8-bit operations using unsigned arithmetic are supported directly. The overflow flag, however, permits the addition and subtraction operation to serve for both unsigned and signed binary integers. Arithmetic can also be performed directly on packed decimal (BCD) representations.

### ADDITION

- INC (increment) adds one to the source operand and puts the result in the operand.
- ADD adds A to the source operand and returns the result to A.
- ADDC (add with Carry) adds A and the source operand, then adds one (1) if CY is set, and puts the result in A.
- DA (decimal-add-adjust for BCD addition) corrects the sum which results from the binary addition of two two-digit decimal operands. The packed decimal sum formed by DA is returned to A. CY is set if the BCD result is greater than 99; otherwise, it is cleared.

### SUBTRACTION

- SUBB (subtract with borrow) subtracts the second source operand from the first operand (the accumulator), subtracts one (1) if CY is set and returns the result to A.
- DEC (decrement) subtracts one (1) from the source operand and returns the result to the operand.

**MULTIPLICATION**

- MUL performs an unsigned multiplication of the A register by the B register, returning a double-byte result. A receives the low-order byte, B receives the high-order byte. OV is cleared if the top half of the result is zero and is set if it is non-zero. CY is cleared. AC is unaffected.

**DIVISION**

- DIV performs an unsigned division of the A register by the B register and returns the integer quotient to A and returns the fractional remainder to the B register. Division by zero leaves indeterminate data in registers A and B and sets OV; otherwise OV is cleared. CY is cleared. AC is unaffected.

Unless otherwise stated in the above descriptions, the flags of PSW are affected as follows:

- CY is set if the operation causes a carry to or from the resulting high-order bit. Otherwise CY is cleared.
- AC is set if the operation results in a carry from the low-order four bits of the result (during addition), or a borrow from the high-order bits to the low-order bits (during subtraction); otherwise AC is cleared.
- OV is set if the operation results in a carry to the high-order bit of the result but not a carry from the high-order bit, or vice versa; otherwise OV is cleared. OV is used in two's-complement arithmetic, because it is set when the signed result cannot be represented in 8 bits.
- P is set if the modulo 2 sum of the eight bits in the accumulator is 1 (odd parity); otherwise P is cleared (even parity). When a value is written to the PSW register, the P bit remains unchanged, as it always reflects the parity of A.20M

**Logic**

The 8051 performs basic logic operations on both bit and byte operands.

**SINGLE-OPERAND OPERATIONS**

- CLR sets A or any directly addressable bit to zero (0).
- SETB sets any directly addressable bit to one (1).
- CPL is used to compliment the contents of the A register without affecting any flags, or any directly addressable bit location.

- RL, RLC, RR, RRC, SWAP are the five rotate operations that can be performed on A. RL, rotate left, RR rotate right, RLC, rotate left through C, RRC rotate right through C, and SWAP, rotate left four. For RLC and RRC the CY flag becomes equal to the last bit rotated out. SWAP rotates A left four places to exchange bits 3 through 0 with bits 7 through 4.

**TWO-OPERAND OPERATIONS**

- ANL performs bitwise logical AND of two source operands (for both bit and byte operands) and returns the result to the location of the first operand.
- ORL performs bitwise logical OR of two source operands (for both bit and byte operands) and returns the result to the location of the first operand.
- XRL performs bitwise logical XOR of two source operands (byte operands) and returns the result to the location of the first operand.

**Control Transfer**

There are three classes of control transfer operations; unconditional calls, returns and jumps; conditional jumps; and interrupts. All control transfer operations cause, some upon a specific condition, the program execution to continue at a non-sequential location in program memory.

**UNCONDITIONAL CALLS, RETURNS AND JUMPS**

Unconditional calls, returns and jumps transfer control from the current value of the Program Counter to the target address. Both direct and indirect transfers are supported.

- ACALL and LCALL push the address of the next instruction onto the stack and then transfer control to the target address. ACALL is a 2-byte instruction used when the target address is in the current 2K page. LCALL is a 3-byte instruction that addresses the full 64K program space. In ACALL, immediate data (i.e., an 11 bit address field) is concatenated to the five most significant bits of the PC (which is pointing to the next instruction). If ACALL is in the last 2 bytes of a 2K page then the call will be made to the next page since the PC will have been incremented to the next instruction prior to execution.

- RET transfers control to the return address saved on the stack by a previous call operation and decrements the SP register by two (2) to adjust the SP for the popped address.

- AJMP, LJMP and SJMP transfer control to the target operand. The operation of AJMP and LJMP are analogous to ACALL and LCALL. The SJMP (short jump) instruction provides for transfers within a 256-byte range centered about the starting address of the next instruction (−128 to +127).

- JMP @A+DPTR performs a jump relative to the DPTR register. The operand in A is used as the offset (0-255) to the address in the DPTR register. Thus, the effective destination for a jump can be anywhere in the Program Memory space.

### CONDITIONAL JUMPS

Conditional jumps perform a jump contingent upon a specific condition. The destination will be within a 256-byte range centered about the starting address of the next instruction (−128 to +127).

- JZ performs a jump if the accumulator is zero.
- JNZ performs a jump if the accumulator is not zero.
- JC performs a jump if the carry flag is set.
- JNC performs a jump if the carry flag is not set.
- JB performs a jump if the Direct Addressed bit is set.
- JNB performs a jump if the Direct Addressed bit is not set.
- JBC performs a jump if the Direct Addressed bit is set and then clears the Direct Addressed bit.
- CJNE compares the first operand to the second operand and performs a jump if they are not equal. CY is set if the first operand is less than the second operand; otherwise it is cleared. Comparisons can be made between directly addressable bytes in Internal Data Memory or

between an immediate value and either A, a register in the selected Register Bank, or a Register-Indirect addressed byte of Internal RAM.

- DJNZ decrements the source operand and returns the result to the operand. A jump is performed if the result is not zero. The source operand of the DJNZ instruction may be any byte in the Internal Data Memory. Either Direct or Register Addressing may be used to address the source operand.

### INTERRUPT RETURNS

- RETI transfers control as does RET, but additionally enables interrupts of the current priority level.

### INSTRUCTION DEFINITIONS

Each of the 51 basic MCS-51 operations, ordered alphabetically according to the operation mnemonic are described beginning page 8-33.

A brief example of how the instruction might be used is given as well as its effect on the PSW flags. The number of bytes and machine cycles required, the binary machine-language encoding, and a symbolic description or restatement of the function is also provided.

Note: Only the carry, auxiliary-carry, and overflow flags are discussed. The parity bit is computed after every instruction cycle that alters the accumulator. Similarly, instructions which alter directly addressed registers could affect the other status flags if the instruction is applied to the PSW. Status flags can also be modified by bit-manipulation.

For details on the MCS-51 assembler, ASM51, refer to the MCS-51 Macro Assembler User's Guide, publication number 9800937.

Table 8-10 summarizes the MCS-51 instruction set.



Table 8-10. 8051 Instruction Set Summary

|  |             |                    |             |   |  |  |  |
|--|-------------|--------------------|-------------|---|--|--|--|
| Interrupt Response Time: To finish execution of current instruction, respond to the interrupt request, push the PC and to vector to the first instruction of the interrupt service program requires 38 to 86 oscillator periods (3 to 7 $\mu$ s @ 12 MHz). |             |                    |             | Notes on instruction set and addressing modes:              |  |  |  |
| <b>INSTRUCTIONS THAT AFFECT FLAG SETTINGS<sup>1</sup></b>  |             |                    |             | Rn —Register R7–R0 of the currently selected Register Bank. |  |  |  |
| <b>INSTRUCTION</b>   | <b>FLAG</b> | <b>INSTRUCTION</b> | <b>FLAG</b> | <b>direct</b>   | —8-bit internal data location's address. This could be an Internal Data RAM location (0–127) or a SFR [i.e., I/O port, control register, status register, etc. (128–255)]. |  |  |
|  | C OV AC     |                    | C OV AC     | <b>@Ri</b>  | —8-bit internal data RAM location (0–255) addressed indirectly through register R1 or R0.  |  |  |
| ADD  | X X X       | CLR C              | O           | <b>#data</b>  | —8-bit constant included in instruction.   |  |  |
| ADDC   | X X X       | CPLC               | X           | <b>#data 16</b>   | —16-bit constant included in instruction   |  |  |
| SUBB   | X X X       | ANL C,bit          | X           | <b>addr 16</b>  | —16-bit destination address. Used by LCALL & LJMP. A branch can be anywhere within the 64K-byte Program Memory address space.  |  |  |
| MUL  | O X         | ANL C,/bit         | X           | <b>addr 11</b>  | —11-bit destination address. Used by ACALL & AJMP. The branch will be within the same 2K-byte page of program memory as the first byte of the following instruction.       |  |  |
| DIV  | O X         | ORL C,bit          | X           | <b>rel</b>  | —Signed (two's complement) 8-bit offset byte. Used by SJMP and all conditional jumps. Range is –128 to +127 bytes relative to first byte of the following instruction.     |  |  |
| DA   | X           | ORL C,bit          | X           | <b>bit</b>  | —Direct Addressed bit in Internal Data RAM or Special Function Register.   |  |  |
| RRC  | X           | MOV C,bit          | X           | <b>*</b>  | —New operation not provided by 8048AH/8049AH.  |  |  |
| RLC  | X           | CJNE               | X           |   |  |  |  |
| SETBC  | 1           |                    |             |   |  |  |  |

<sup>1</sup>Note that operations on SFR byte address 208 or bit addresses 209-215 (i.e., the PSW or bits in the PSW) will also affect flag settings.

<sup>1</sup>Note that operations on SFR byte address 208 or bit addresses 209-215 (i.e., the PSW or bits in the PSW) will also affect flag settings.

| ARITHMETIC OPERATIONS |  |      |                   |
|-----------------------|--|------|-------------------|
| Mnemonic              | Description                                | Byte | Oscillator Period |
| ADD A, Rn             | Add register to Accumulator                | 1    | 12                |
| ADD A, direct         | Add direct byte to Accumulator             | 2    | 12                |
| ADD A, @Ri            | Add indirect RAM to Accumulator            | 1    | 12                |
| ADD A, #data          | Add immediate data to Accumulator          | 2    | 12                |
| ADDC A, Rn            | Add register to Accumulator with Carry     | 1    | 12                |
| ADDC A, direct        | Add direct byte to Accumulator with Carry  | 2    | 12                |
| ADDC A, @Ri           | Add indirect RAM to Accumulator with Carry | 1    | 12                |
| ADDC A, #data         | Add immediate data to Acc with Carry       | 2    | 12                |
| SUBB A, Rn            | Subtract register from Acc with borrow     | 1    | 12                |
| SUBB A, direct        | Subtract direct byte from Acc with borrow  | 2    | 12                |

| ARITHMETIC OPERATIONS Cont. |  |      |                   |
|-----------------------------|--|------|-------------------|
| Mnemonic                    | Description                                  | Byte | Oscillator Period |
| SUBB A, @Ri                 | Subtract indirect RAM from Acc with borrow   | 1    | 12                |
| SUBB A, #data               | Subtract immediate data from Acc with borrow | 2    | 12                |
| INC A                       | Increment Accumulator                        | 1    | 12                |
| INC Rn                      | Increment register                           | 1    | 12                |
| INC direct                  | Increment direct byte                        | 2    | 12                |
| INC @Ri                     | Increment indirect RAM                       | 1    | 12                |
| DEC A                       | Decrement Accumulator                        | 1    | 12                |
| DEC Rn                      | Decrement Register                           | 1    | 12                |
| DEC direct                  | Decrement direct byte                        | 2    | 12                |
| DEC @Ri                     | Decrement indirect RAM                       | 1    | 12                |
| INC DPTR                    | Increment Data Pointer                       | 1    | 24                |
| MUL AB                      | Multiply A & B                               | 1    | 48                |
| DIV AB                      | Divide A by B                                | 1    | 48                |
| DA A                        | Decimal Adjust Accumulator                   | 1    | 12                |

All mnemonics copyrighted ©Intel Corporation 1980

Table 8-10. 8051 Instruction Set Summary (Continued)

| LOGICAL OPERATIONS |   |      |                   | LOGICAL OPERATIONS Cont. |  |      |                   |
|--------------------|---|------|-------------------|--------------------------|--|------|-------------------|
| Mnemonic           | Description                             | Byte | Oscillator Period | Mnemonic                 | Description                                | Byte | Oscillator Period |
| ANL A,Rn           | AND register to Accumulator             | 1    | 12                | XRL A,@Ri                | Exclusive-OR indirect RAM to Accumulator   | 1    | 12                |
| ANL A,direct       | AND direct byte to Accumulator          | 2    | 12                | XRL A,#data              | Exclusive-OR immediate data to Accumulator | 2    | 12                |
| ANL A,@Ri          | AND indirect RAM to Accumulator         | 1    | 12                | XRL direct,A             | Exclusive-OR Accumulator to direct byte    | 2    | 12                |
| ANL A,#data        | AND immediate data to Accumulator       | 2    | 12                | XRL direct,#data         | Exclusive-OR immediate data to direct byte | 3    | 24                |
| ANL direct,A       | AND Accumulator to direct byte          | 2    | 12                | CLR A                    | Clear Accumulator                          | 1    | 12                |
| ANL direct,#data   | AND immediate data to direct byte       | 3    | 24                | CPL A                    | Complement Accumulator                     | 1    | 12                |
| ORL A,Rn           | OR register to Accumulator              | 1    | 12                | RL A                     | Rotate Accumulator Left                    | 1    | 12                |
| ORL A,direct       | OR direct byte to Accumulator           | 2    | 12                | RLC A                    | Rotate Accumulator Left through the Carry  | 1    | 12                |
| ORL A,@Ri          | OR indirect RAM to Accumulator          | 1    | 12                | RR A                     | Rotate Accumulator Right                   | 1    | 12                |
| ORL A,#data        | OR immediate data to Accumulator        | 2    | 12                | RRC A                    | Rotate Accumulator Right through the Carry | 1    | 12                |
| ORL direct,A       | OR Accumulator to direct byte           | 2    | 12                | SWAP A                   | Swap nibbles within the Accumulator        | 1    | 12                |
| ORL direct,#data   | OR immediate data to direct byte        | 3    | 24                |                          |  |      |                   |
| XRL A,Rn           | Exclusive-OR register to Accumulator    | 1    | 12                |                          |  |      |                   |
| XRL A,direct       | Exclusive-OR direct byte to Accumulator | 2    | 12                |                          |  |      |                   |

All mnemonics copyrighted ©Intel Corporation 1980

Table 8-10. 8051 Instruction Set Summary (Continued)

| DATA TRANSFER |               |                                     |      |                   | DATA TRANSFER Cont. |              |  |      |                   |
|---------------|---------------|-------------------------------------|------|-------------------|---------------------|--------------|--|------|-------------------|
|               | Mnemonic      | Description                         | Byte | Oscillator Period |                     | Mnemonic     | Description                                    | Byte | Oscillator Period |
| MOV           | A,Rn          | Move register to Accumulator        | 1    | 12                | MOV                 | DPTR,#data16 | Load Data Pointer with a 16-bit constant       | 3    | 24                |
| MOV           | A,direct      | Move direct byte to Accumulator     | 2    | 12                | MOVC                | A,@A+DPTR    | Move Code byte relative to DPTR to Acc         | 1    | 24                |
| MOV           | A,@Ri         | Move indirect RAM to Accumulator    | 1    | 12                | MOVC                | A,@A+PC      | Move Code byte relative to PC to Acc           | 1    | 24                |
| MOV           | A,#data       | Move immediate data to Accumulator  | 2    | 12                | MOVX                | A,@Ri        | Move External RAM (8-bit addr) to Acc          | 1    | 24                |
| MOV           | Rn,A          | Move Accumulator to register        | 1    | 12                | MOVX                | A,@DPTR      | Move External RAM (16-bit addr) to Acc         | 1    | 24                |
| MOV           | Rn,direct     | Move direct byte to register        | 2    | 24                | MOVX                | @Ri,A        | Move Acc to External RAM (8-bit addr)          | 1    | 24                |
| MOV           | Rn,#data      | Move immediate data to register     | 2    | 12                | MOVX                | @DPTR,A      | Move Acc to External RAM (16-bit addr)         | 1    | 24                |
| MOV           | direct,A      | Move Accumulator to direct byte     | 2    | 12                | PUSH                | direct       | Push direct byte onto stack                    | 2    | 24                |
| MOV           | direct,Rn     | Move register to direct byte        | 2    | 24                | POP                 | direct       | Pop direct byte from stack                     | 2    | 24                |
| MOV           | direct,direct | Move direct byte to direct          | 3    | 24                | XCH                 | A,Rn         | Exchange register with Accumulator             | 1    | 12                |
| MOV           | direct,@Ri    | Move indirect RAM to direct byte    | 2    | 24                | XCH                 | A,direct     | Exchange direct byte with Accumulator          | 2    | 12                |
| MOV           | direct,#data  | Move immediate data to direct byte  | 3    | 24                | XCH                 | A,@Ri        | Exchange indirect RAM with Accumulator         | 1    | 12                |
| MOV           | @Ri,A         | Move Accumulator to indirect RAM    | 1    | 12                | XCHD                | A,@Ri        | Exchange low-order Digit indirect RAM with Acc | 1    | 12                |
| MOV           | @Ri,direct    | Move direct byte to indirect RAM    | 2    | 24                |                     |              |  |      |                   |
| MOV           | @Ri,#data     | Move immediate data to indirect RAM | 2    | 12                |                     |              |  |      |                   |

All mnemonics copyrighted ©Intel Corporation 1980

Table 8-10. 8051 Instruction Set Summary (Continued)

| BOOLEAN VARIABLE MANIPULATION |                                       |      |                   |  | PROGRAM BRANCHING Cont. |   |      |                   |  |
|-------------------------------|---------------------------------------|------|-------------------|--|-------------------------|---|------|-------------------|--|
| Mnemonic                      | Description                           | Byte | Oscillator Period |  | Mnemonic                | Description   | Byte | Oscillator Period |  |
| CLR C                         | Clear Carry                           | 1    | 12                |  | RETI                    | Return from interrupt                               | 1    | 24                |  |
| CLR bit                       | Clear direct bit                      | 2    | 12                |  | AJMP addr11             | Absolute Jump                                       | 2    | 24                |  |
| SETB C                        | Set Carry                             | 1    | 12                |  | LJMP addr16             | Long Jump   | 3    | 24                |  |
| SETB bit                      | Set direct bit                        | 2    | 12                |  | SJMP rel                | Short Jump (relative addr)                          | 2    | 24                |  |
| CPL                           | Complement Carry                      | 1    | 12                |  | JMP @A+DPTR             | Jump indirect relative to the DPTR                  | 1    | 24                |  |
| CPL bit                       | Complement direct bit                 | 2    | 12                |  | JZ rel                  | Jump if Accumulator is Zero                         | 2    | 24                |  |
| ANL C,bit                     | AND direct bit to Carry               | 2    | 24                |  | JNZ rel                 | Jump if Accumulator is Not Zero                     | 2    | 24                |  |
| ANL C,/bit                    | AND complement of direct bit to Carry | 2    | 24                |  | CJNE A,direct,rel       | Compare direct byte to Acc and Jump if Not Equal    | 3    | 24                |  |
| ORL C,bit                     | OR direct bit to Carry                | 2    | 24                |  | CJNE A,#data,rel        | Compare immediate to Acc and Jump if Not Equal      | 3    | 24                |  |
| ORL C,/bit                    | OR complement of direct bit to Carry  | 2    | 24                |  | CJNE Rn,#data,rel       | Compare immediate to register and Jump If Not Equal | 3    | 24                |  |
| MOV C,bit                     | Move direct bit to Carry              | 2    | 12                |  | CJNE @Ri,#data,rel      | Compare immediate to indirect and Jump if Not Equal | 3    | 24                |  |
| MOV bit,C                     | Move Carry to direct bit              | 2    | 24                |  | DJNZ Rn,rel             | Decrement register and Jump if Not Zero             | 2    | 24                |  |
| JC rel                        | Jump if Carry is set                  | 2    | 24                |  | DJNZ direct,rel         | Decrement direct byte and Jump if Not Zero          | 3    | 24                |  |
| JNC rel                       | Jump if Carry not set                 | 2    | 24                |  | NOP                     | No Operation  | 1    | 12                |  |
| JB bit,rel                    | Jump if direct Bit is set             | 3    | 24                |  |                         |   |      |                   |  |
| JNB bit,rel                   | Jump if direct Bit is Not set         | 3    | 24                |  |                         |   |      |                   |  |
| JBC bit,rel                   | Jump if direct Bit is set & clear bit | 3    | 24                |  |                         |   |      |                   |  |

| PROGRAM BRANCHING |                          |      |                   |  |
|-------------------|--------------------------|------|-------------------|--|
| Mnemonic          | Description              | Byte | Oscillator Period |  |
| ACALL addr11      | Absolute Subroutine Call | 2    | 24                |  |
| LCALL addr16      | Long Subroutine Call     | 3    | 24                |  |
| RET               | Return from Subroutine   | 1    | 24                |  |

All mnemonics copyrighted ©Intel Corporation 1980



## INSTRUCTION DEFINITIONS

ACALL      addr11

**Function:** Absolute Call

**Description:** ACALL unconditionally calls a subroutine located at the indicated address. The instruction increments the PC twice to obtain the address of the following instruction, then pushes the 16-bit result onto the stack (low-order byte first) and increments the stack pointer twice. The destination address is obtained by successively concatenating the five high-order bits of the incremented PC, opcode bits 7-5, and the second byte of the instruction. The subroutine called must therefore start within the same 2K block of the program memory as the first byte of the instruction following ACALL. No flags are affected.

**Example:** Initially SP equals 07H. The label "SUBRTN" is at program memory location 0345H. After executing the instruction,

ACALL      SUBRTN

at location 0123H, SP will contain 09H, internal RAM locations 08H and 09H will contain 25H and 01H, respectively, and the PC will contain 0345H.

**Bytes:** 2**Cycles:** 2**Encoding:**

|     |    |    |   |   |   |   |   |    |    |    |    |    |    |    |    |
|-----|----|----|---|---|---|---|---|----|----|----|----|----|----|----|----|
| a10 | a9 | a8 | 1 | 0 | 0 | 0 | 1 | a7 | a6 | a5 | a4 | a3 | a2 | a1 | a0 |
|-----|----|----|---|---|---|---|---|----|----|----|----|----|----|----|----|

**Operation:**

ACALL

 $(PC) \leftarrow (PC) + 2$  $(SP) \leftarrow (SP) + 1$  $((SP)) \leftarrow (PC_{7-0})$  $(SP) \leftarrow (SP) + 1$  $((SP)) \leftarrow (PC_{15-8})$  $(PC_{10-0}) \leftarrow \text{page address}$

## INSTRUCTION DEFINITIONS

**ADD     A,<src-byte>****Function:** Add

**Description:** ADD adds the byte variable indicated to the accumulator, leaving the result in the accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred.

OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out of bit 7 but not bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands.

Four source operand addressing modes are allowed: register, direct, register-indirect, or immediate.

**Example:** The accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B). The instruction,

ADD     A,R0

will leave 6DH (01101101B) in the accumulator with the AC flag cleared and both the carry flag and OV set to 1.

**ADD   A,Rn****Bytes:** 1**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 | r | r | r |
|---|---|---|---|---|---|---|---|

**Operation:** ADD  
 $(A) \leftarrow (A) + (Rn)$

**ADD   A,direct****Bytes:** 2**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|

|                |
|----------------|
| direct address |
|----------------|

**Operation:** ADD  
 $(A) \leftarrow (A) + (\text{direct})$

# ADD A,@Ri

Bytes: 1

Cycles: 1

Encoding:

0 0 1 0 0 1 1 i

Operation:

ADD  
(A) ← (A) + ((Ri))

# ADD A,#data

Bytes: 2

Cycles: 1

Encoding:

0 0 1 0 0 1 0 0

immediate data

Operation:

ADD  
(A) ← (A) + #data

## ADDC A, <src-byte>

**Function:** Add with Carry

**Description:** ADDC simultaneously adds the byte variable indicated, the carry flag and the accumulator contents, leaving the result in the accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred.

OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carryout of bit 7 but not out of bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands or a positive sum from two negative operands.

Four source operand addressing modes are allowed: register, direct, register-indirect, or immediate.

**Example:** The accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B) with the carry flag set. The instruction,

ADDC A,R0

will leave 6EH (01101110B) in the accumulator with AC cleared and both the carry flag and OV set to 1.

### ADDC A,Rn

**Bytes:** 1

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 | r | r | r |
|---|---|---|---|---|---|---|---|

**Operation:** ADDC  
 $(A) \leftarrow (A) + (C) + (R_n)$

### ADDC A,direct

**Bytes:** 2

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|

|                |
|----------------|
| direct address |
|----------------|

**Operation:** ADDC  
 $(A) \leftarrow (A) + (C) + (\text{direct})$



**ADDC A,@Ri****Bytes:** 1**Cycles:** 1**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 1 | 1 | i |
|---|---|---|---|---|---|---|---|

**Operation:**

ADDC

 $(A) \leftarrow (A) + (C) + ((R_i))$ **ADDC A,#data****Bytes:** 2**Cycles:** 1**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|

immediate data

**Operation:**

ADDC

 $(A) \leftarrow (A) + (C) + \#data$ **AJMP addr11****Function:**

Absolute Jump

**Description:**

AJMP transfers program execution to the indicated address, which is formed at run-time by concatenating the high-order five bits of the PC (*after* incrementing the PC twice), opcode bits 7-5, and the second byte of the instruction. The destination must therefore be within the same 2K block of program memory as the first byte of the instruction following AJMP.

**Example:**

The label "JMPADR" is at program memory location 0123H. The instruction, AJMP JMPADR

is at location 0345H and will load the PC with 0123H.

**Bytes:** 2**Cycles:** 2**Encoding:**

|     |    |    |   |   |   |   |   |
|-----|----|----|---|---|---|---|---|
| a10 | a9 | a8 | 0 | 0 | 0 | 0 | 1 |
|-----|----|----|---|---|---|---|---|

|    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|
| a7 | a6 | a5 | a4 | a3 | a2 | a1 | a0 |
|----|----|----|----|----|----|----|----|

**Operation:**

AJMP

 $(PC) \leftarrow (PC) + 2$  $(PC_{10-0}) \leftarrow \text{page address}$

**ANL**    <dest-byte> , <src-byte>**Function:** Logical-AND for byte variables**Description:** ANL performs the bitwise logical-AND operation between the variables indicated and stores the results in the destination variable. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the accumulator or immediate data.

*Note:* When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

**Example:** If the accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B) then the instruction,

```
ANL    A,R0
```

will leave 41H (01000001B) in the accumulator.

When the destination is a directly addressed byte, this instruction will clear combinations of bits in any RAM location or hardware register. The mask byte determining the pattern of bits to be cleared would either be a constant contained in the instruction or a value computed in the accumulator at run-time.

The instruction,

```
ANL    P1,#01110011B
```

will clear bits 7, 3, and 2 of output port 1.

**ANL A,Rn****Bytes:** 1**Cycles:** 1**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 1 | r | r | r |
|---|---|---|---|---|---|---|---|

**Operation:**

ANL

$$(A) \leftarrow (A) \wedge (Rn)$$

**ANL A, direct****Bytes:** 2**Cycles:** 1**Encoding:**

0 1 0 1 0 1 0 1

direct address

**Operation:**

ANL

 $(A) \leftarrow (A) \wedge (\text{direct})$ **ANL A, @Ri****Bytes:** 1**Cycles:** 1**Encoding:**

0 1 0 1 0 1 1 i

**Operation:**

ANL

 $(A) \leftarrow (A) \wedge ((Ri))$ **ANL A, #data****Bytes:** 2**Cycles:** 1**Encoding:**

0 1 0 1 0 1 0 0

immediate data

**Operation:**

ANL

 $(A) \leftarrow (A) \wedge \#data$ **ANL direct, A****Bytes:** 2**Cycles:** 1**Encoding:**

0 1 0 1 0 0 1 0

direct address

**Operation:**

ANL

 $(\text{direct}) \leftarrow (\text{direct}) \wedge (A)$ **ANL direct, #data****Bytes:** 3**Cycles:** 2**Encoding:**

0 1 0 1 0 0 1 1

direct address

immediate data

**Operation:**

ANL

 $(\text{direct}) \leftarrow (\text{direct}) \wedge \#data$

**ANL C, <src-bit>****Function:** Logical-AND for bit variables**Description:** If the Boolean value of the source bit is a logical 0 then clear the carry flag; otherwise leave the carry flag in its current state. A slash ("/") preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, *but the source bit itself is not affected*. No other flags are affected.**Example:** Only direct bit addressing is allowed for the source operand.

Set the carry flag if, and only if, P1.0 = 1, ACC. 7 = 1, and OV = 0:

```

MOV C,P1.0           ;LOAD CARRY WITH INPUT PIN STATE
ANL C,ACC.7          ;AND CARRY WITH ACCUM. BIT 7
ANL C,/OV             ;AND WITH INVERSE OF OVERFLOW
                      FLAG

```

**ANL C,bit****Bytes:** 2**Cycles:** 2**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|---|

 bit address**Operation:** ANL  
 $(C) \leftarrow (C) \wedge (\text{bit})$ **ANL C,bit****Bytes:** 2**Cycles:** 2**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

 bit address**Operation:** ANL  
 $(C) \leftarrow (C) \wedge \neg (\text{bit})$ **CJNE <dest-byte>, <src-byte>, rel****Function:** Compare and Jump if Not Equal.**Description:** CJNE compares the magnitudes of the first two operands, and branches if their values are not equal. The branch destination is computed by adding the signed relative-displacement in the last instruction byte to the PC, after incrementing the PC to the start of the next instruction. The carry flag is set if the unsigned integer value of <dest-byte> is less than the unsigned integer value of <src-byte>; otherwise, the carry is cleared. Neither operand is affected.



The first two operands allow four addressing mode combinations: the accumulator may be compared with any directly addressed byte or immediate data, and any indirect RAM location or working register can be compared with an immediate constant.

**Example:** The accumulator contains 34H. Register 7 contains 56H. The first instruction in the sequence,

```

                CJNE    R7,#60H, NOT_EQ
;              ...      ...      ; R7 = 60H.
NOT_EQ:        JC      REQ_LOW    ; IF R7 < 60H.
;              ...      ...      ; R7 > 60H.

```

sets the carry flag and branches to the instruction at label NOT\_EQ. By testing the carry flag, this instruction determines whether R7 is greater or less than 60H.

If the data being presented to port 1 is also 34H, then the instruction,

```
WAIT: CJNE A,P1,WAIT
```

clears the carry flag and continues with the next instruction in sequence, since the accumulator does equal the data read from P1. (If some other value was being input on P1, the program will loop at this point until the P1 data changes to 34H.)

**CJNE A,direct,rel**

**Bytes:** 3

**Cycles:** 2

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|

direct address

rel. address

**Operation:**

$(PC) \leftarrow (PC) + 3$

IF (A) <> (direct)

THEN

$(PC) \leftarrow (PC) + \text{relative offset}$

IF (A) < (direct)

THEN

ELSE

$(C) \leftarrow 0$

**CJNE A,#data,rel****Bytes:** 3**Cycles:** 2**Encoding:**

1 0 1 1 0 1 0 0

immediate data

rel. address

**Operation:** $(PC) \leftarrow (PC) + 3$ 

IF (A) &lt;&gt; data

THEN

 $(PC) \leftarrow (PC) + \text{relative offset}$ 

IF (A) &lt; data

THEN

 $(C) \leftarrow 1$ 

ELSE

 $(C) \leftarrow 0$ **CJNE Rn,#data,rel****Bytes:** 3**Cycles:** 2**Encoding:**

1 0 1 1 1 r r r

immediate data

rel. address

**Operation:** $(PC) \leftarrow (PC) + 3$ 

IF (Rn) &lt;&gt; data

THEN

 $(PC) \leftarrow (PC) + \text{relative offset}$ 

IF (Rn) &lt; data

THEN

 $(C) \leftarrow 1$ 

ELSE

 $(C) \leftarrow 0$ **CJNE @Ri,#data,rel****Bytes:** 3**Cycles:** 2**Encoding:**

1 0 1 1 0 1 1 i

immediate data

rel. address

**Operation:** $(PC) \leftarrow (PC) + 3$ 

IF ((Ri)) &lt;&gt; data

THEN

 $(PC) \leftarrow (PC) + \text{relative offset}$ 

IF ((Ri)) &lt; data

THEN

 $(C) \leftarrow 1$ 

ELSE

 $(C) \leftarrow 0$

**CLR A**

**Function:** Clear Accumulator

**Description:** The accumulator is cleared (all bits set to zero). No flags are affected.

**Example:** The accumulator contains 5CH (01011100B). The instruction, CLR A, will leave the accumulator set to 00H (00000000B).

**Bytes:** 1

**Cycles:** 1

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 1 | 1 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
|---|---|---|---|

**Operation:** CLR  
(A) ← 0

**CLR bit**

**Function:** Clear bit

**Description:** The indicated bit is cleared (reset to zero). No other flags are affected. CLR can operate on the carry flag or any directly addressable bit.

**Example:** Port 1 has previously been written with 5DH (01011101B). The instruction, CLR P1.2, will leave the port set to 59H (01011001B).

**CLR C**

**Bytes:** 1

**Cycles:** 1

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 1 | 0 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
|---|---|---|---|

**Operation:** CLR  
(C) ← 0

**CLR bit**

**Bytes:** 2

**Cycles:** 1

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 1 | 0 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 1 | 0 |
|---|---|---|---|

|             |   |   |   |   |
|-------------|---|---|---|---|
| bit address | 1 | 0 | 1 | 1 |
|-------------|---|---|---|---|

**Operation:** CLR  
(bit) ← 0

**CPL A****Function:** Complement Accumulator**Description:** Each bit of the accumulator is logically complemented (one's complement). Bits which previously contained a one are changed to zero and vice-versa. No flags are affected.**Example:** The accumulator contains 5CH (01011100B). The instruction,

CPL A

will leave the accumulator set to 0A3H (10100011B).

**Bytes:** 1**Cycles:** 1**Encoding:**

1 1 1 1 0 1 0 0

**Operation:**

CPL

 $(A) \leftarrow \neg (A)$ **CPL bit****Function:** Complement bit**Description:** The bit variable specified is complemented. A bit which had been a one is changed to zero and vice-versa. No other flags are affected. CLR can operate on the carry or any directly addressable bit.**Example:***Note:* When this instruction is used to modify an output pin, the value used as the original data will be read from the output data latch, *not* the input pin.

Port 1 has previously been written with 5BH (01011101B). The instruction sequence,

CPL P1.1

CPL P1.2

will leave the port set to 5BH (01011011B).

**CPL C****Bytes:** 1**Cycles:** 1**Encoding:**

1 0 1 1 0 0 1 1

**Operation:**

CPL

 $(C) \leftarrow \neg (C)$



**CPL** bit  
**Bytes:** 2  
**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|---|

bit address

**Operation:** CPL  
 (bit) ← ¬(bit)

**DA A**

**Function:** Decimal-adjust Accumulator for Addition

**Description:** DA A adjusts the eight-bit value in the accumulator resulting from the earlier addition of two variables (each in packed-BCD format), producing two four-bit digits. Any ADD or ADDC instruction may have been used to perform the addition.

If accumulator bits 3-0 are greater than nine (xxxx1010-xxxx1111), or if the AC flag is one, six is added to the accumulator producing the proper BCD digit in the low-order nibble. This internal addition would set the carry flag if a carry-out of the low-order four-bit field propagated through all high-order bits, but it would not clear the carry flag otherwise.

If the carry flag is now set, or if the four high-order bits now exceed nine (1010xxxx-1111xxxx), these high-order bits are incremented by six, producing the proper BCD digit in the high-order nibble. Again, this would set the carry flag if there was a carry-out of the high-order bits, but wouldn't clear the carry. The carry flag thus indicates if the sum of the original two BCD variables is greater than 100, allowing multiple precision decimal addition. OV is not affected.

All of this occurs during the one instruction cycle. Essentially, this instruction performs the decimal conversion by adding 00H, 06H, 60H, or 66H to the accumulator, depending on initial accumulator and PSW conditions.

*Note:* DA A cannot simply convert a hexadecimal number in the accumulator to BCD notation, nor does DA A apply to decimal subtraction.

**Example:** The accumulator holds the value 56H (01010110B) representing the packed BCD digits of the decimal number 56. Register 3 contains the value 67H (01100111B) representing the packed BCD digits of the decimal number 67. The carry flag is set. The instruction sequence,

```
ADDC  A,R3
DA     A
```

will first perform a standard two's-complement binary addition, resulting in the value 0BEH (10111110) in the accumulator. The carry and auxiliary carry flags will be cleared.

The Decimal Adjust instruction will then alter the accumulator to the value 24H (00100100B), indicating the packed BCD digits of the decimal number 24, the low-order two digits of the decimal sum of 56, 67, and the carry-in. The carry flag will be set by the Decimal Adjust instruction, indicating that a decimal overflow occurred. The true sum 56, 67, and 1 is 124.

BCD variables can be incremented or decremented by adding 01H or 99H. If the accumulator initially holds 30H (representing the digits of 30 decimal), then the instruction sequence,

```
ADD  A,#99H
DA   A
```

will leave the carry set and 29H in the accumulator, since  $30 + 99 = 129$ . The low-order byte of the sum can be interpreted to mean  $30 - 1 = 29$ .

Bytes: 1  
Cycles: 1

Encoding: 1 1 0 1 0 1 0 0

Operation: DA

-contents of Accumulator are BCD

IF  $[(A3-0) > 9] \vee [(AC) = 1]$   
THEN  $(A3-0) \leftarrow (A3-0) + 6$

AND

IF  $[(A7-4) > 9] \vee [(C) = 1]$   
THEN  $(A7-4) \leftarrow (A7-4) + 6$

# DEC byte

**Function:** Decrement

**Description:** The variable indicated is decremented by 1. An original value of 00H will underflow to 0FFH. No flags are affected. Four operand addressing modes are allowed: accumulator, register, direct, or register-indirect.

*Note:* When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

**Example:** Register 0 contains 7FH (01111111B). Internal RAM locations 7EH and 7FH contain 00H and 40H, respectively. The instruction sequence,

```
DEC @R0
DEC R0
DEC @R0
```

will leave register 0 set to 7EH and internal RAM locations 7EH and 7FH set to 0FFH and 3FH.

# DEC A

**Bytes:** 1

**Cycles:** 1

**Encoding:** 0 0 0 1 0 1 0 0

**Operation:** DEC

$(A) \leftarrow (A) - 1$

# DEC Rn

**Bytes:** 1

**Cycles:** 1

**Encoding:** 0 0 0 1 1 r r r

**Operation:** DEC

$(Rn) \leftarrow (Rn) - 1$

|                     |   |
|---------------------|---|
| <b>Function:</b>    | Divide  |
| <b>Description:</b> | DIV AB divides the unsigned eight-bit integer in the accumulator by the unsigned eight-bit integer in register B. The accumulator receives the integer part of the quotient; register B receives the integer remainder. The carry and OV flags will be cleared. |

**Exception:** if B had originally contained 00H, the values returned in the accumulator and B-register will be undefined and the overflow flag will be set. The carry flag is cleared in any case.

**Example:** The accumulator contains 251 (0FBH or 11111011B) and B contains 18 (12H or 00010010B). The instruction,

DIV AB

will leave 13 in the accumulator (0DH or 00001101B) and the value 17 (11H or 00010001B) in B, since  $251 = (13 \times 18) + 17$ . Carry and OV will both be cleared.

**Bytes:** 1  
**Cycles:** 4

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|

**Operation:** DIV  
(A)15-8 ← (A) / (B)  
(B)7-0



**DJNZ** <byte>, <rel-addr>**Function:** Decrement and Jump if Not Zero**Description:** DJNZ decrements the location indicated by 1, and branches to the address indicated by the second operand if the resulting value is not zero. An original value of 00H will underflow to 0FFH. No flags are affected. The branch destination would be computed by adding the signed relative-displacement value in the last instruction byte to the PC, after incrementing the PC to the first byte of the following instruction.

The location decremented may be a register or directly addressed byte.

*Note:* When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.**Example:** Internal RAM locations 40H, 50H, and 60H contain the values 01H, 70H, and 15H, respectively. The instruction sequence,

```

DJNZ 40H, LABEL_1
DJNZ 50H, LABEL_2
DJNZ 60H, LABEL_3

```

will cause a jump to the instruction at label LABEL\_2 with the values 00H, 6FH, and 15H in the three RAM locations. The first jump was *not* taken because the result was zero.

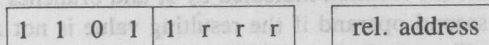
This instruction provides a simple way of executing a program loop a given number of times, or for adding a moderate time delay (from 2 to 512 machine cycles) with a single instruction. The instruction sequence,

```

MOV R2, #8
TOGGLE: CPL P1.7
        DJNZ R2, TOGGLE

```

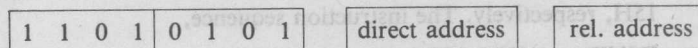
will toggle P1.7 eight times, causing four output pulses to appear at bit 7 of output port 1. Each pulse will last three machine cycles; two for DJNZ and one to alter the pin.

**DJNZ Rn,rel****Bytes:** 2**Cycles:** 2**Encoding:****Operation:**

DJNZ

 $(PC) \leftarrow (PC) + 2$  $(Rn) \leftarrow (Rn) - 1$ IF  $(Rn) > 0$  or  $(Rn) < 0$ 

THEN

 $(PC) \leftarrow (PC) + \text{rel}$ **DJNZ direct,rel****Bytes:** 3**Cycles:** 2**Encoding:****Operation:**

DJNZ

 $(PC) \leftarrow (PC) + 2$  $(\text{direct}) \leftarrow (\text{direct}) - 1$ IF  $(\text{direct}) > 0$  or  $(\text{direct}) < 0$ 

THEN

 $(PC) \leftarrow (PC) + \text{rel}$ **INC <byte>****Function:**

Increment

**Description:**

INC increments the indicated variable by 1. An original value of 0FFH will overflow to 00H. No flags are affected. Three addressing modes are allowed: register, direct, or register-indirect.

*Note:* When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

**Example:** Register 0 contains 7EH (01111110B). Internal RAM locations 7EH and 7FH contain 0FFH and 40H, respectively. The instruction sequence,

```
INC @R0
INC R0
INC @R0
```

will leave register 0 set to 7FH and internal RAM locations 7EH and 7FH holding (respectively) 00H and 41H.

**INC A**

Bytes: 1

Cycles: 1

Encoding:

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|

Operation:

INC

$(A) \leftarrow (A) + 1$

**INC Rn**

Bytes: 1

Cycles: 1

Encoding:

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | r | r | r |
|---|---|---|---|---|---|---|---|

Operation:

INC

$(Rn) \leftarrow (Rn) + 1$

**INC direct**

Bytes: 2

Cycles: 1

Encoding:

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|

direct address

Operation:

INC

$(\text{direct}) \leftarrow (\text{direct}) + 1$

# INC @Ri

Bytes:

1

Cycles:

1

Encoding:

0 0 0 0 0 0 1 1 i

Operation:

INC  
((Ri)) ← ((Ri)) + 1

# INC DPTR

Function:

Increment Data Pointer

Description:

Increment the 16-bit data pointer by 1. A 16-bit increment (modulo 2<sup>16</sup>) is performed; an overflow of the low-order byte of the data pointer (DPL) from 0FFH to 00H will increment the high-order byte (DPH). No flags are affected.

Example:

This is the only 16-bit register which can be incremented.  
Registers DPH and DPL contain 12H and 0FEH, respectively. The instruction sequence,

INC DPTR

INC DPTR

INC DPTR

will change DPH and DPL to 13H and 01H.

Bytes:

1

Cycles:

2

Encoding:

1 0 1 0 0 0 1 1

Operation:

INC  
(DPTR) ← (DPTR) + 1



**JB     bit,rel****Function:** Jump if Bit set**Description:** If the indicated bit is a one, jump to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. *The bit tested is not modified.* No flags are affected.**Example:** The data present at input port 1 is 11001010B. The accumulator holds 56 (01010110B). The instruction sequence,

```
JB    P1.2,LABEL1
JB    ACC.2,LABEL2
```

will cause program execution to branch to the instruction at label LABEL2.

**Bytes:** 3**Cycles:** 2

|                  |         |         |             |              |
|------------------|---------|---------|-------------|--------------|
| <b>Encoding:</b> | 0 0 1 0 | 0 0 0 0 | bit address | rel. address |
|------------------|---------|---------|-------------|--------------|

**Operation:** JB  
 $(PC) \leftarrow (PC) + 3$   
 IF (bit) = 1  
 THEN  
 $(PC) \leftarrow (PC) + \text{rel}$

**JBC    bit,rel****Function:** Jump if Bit is set and Clear bit**Description:** If the indicated bit is one, branch to the address indicated; otherwise proceed with the next instruction. *The bit will not be cleared if it is already a zero.* The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. No flags are affected.**Example:** The accumulator holds 56H (01010110B). The instruction sequence,

```
JBC   ACC.3,LABEL1
JBC   ACC.2,LABEL2
```

will cause program execution to continue at the instruction identified by the label LABEL2, with the accumulator modified to 52H (01010010B).

Bytes: 3  
Cycles: 2

Encoding: 0 0 0 1 0 0 0 0 bit address rel. address

Operation:

JBC

$(PC) \leftarrow (PC) + 3$

IF (bit) = 1

THEN

$(bit) \leftarrow 0$

$(PC) \leftarrow (PC) + rel$

JC rel

Function:

Jump if Carry is set

Description:

If the carry flag is set, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. No flags are affected.

Example:

The carry flag is cleared. The instruction sequence,

JC LABEL1

CPL C

JC LABEL2

will set the carry and cause program execution to continue at the instruction identified by the label LABEL2.

Bytes:

2

Cycles:

2

Encoding:

0 1 0 0 0 0 0 0 rel. address

Operation:

JC

$(PC) \leftarrow (PC) + 2$

IF (C) = 1

THEN

$(PC) \leftarrow (PC) + rel$

# **JMP @A + DPTR**

**Function:** Jump indirect

**Description:** Add the eight-bit unsigned contents of the accumulator with the sixteen-bit data pointer, and load the resulting sum to the program counter. This will be the address for subsequent instruction fetches. Sixteen-bit addition is performed (modulo 216): a carry-out from the low-order eight bits propagates through the higher-order bits. Neither the accumulator nor the data pointer is altered. No flags are affected.

**Example:** An even number from 0 to 6 is in the accumulator. The following sequence of instructions will branch to one of four AJMP instructions in a jump table starting at JMP\_\_TBL:

```

MOV     DPTR,#JMP__TBL
JMP     @A + DPTR
JMP__TBL: AJMP  LABEL0
          AJMP  LABEL1
          AJMP  LABEL2
          AJMP  LABEL3
  
```

If the accumulator equals 04H when starting this sequence, execution will jump to label LABEL2. Remember that AJMP is a two-byte instruction, so the jump instructions start at every other address.

**Bytes:** 1  
**Cycles:** 2

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|

**Operation:** JMP  
(PC) ← (A) + (DPTR)

**JNB     bit,rel****Function:** Jump if Bit Not set**Description:** If the indicated bit is a zero, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. *The bit tested is not modified.* No flags are affected.**Example:** The data present at input port 1 is 11001010B. The accumulator holds 56H (01010110B). The instruction sequence,

```
JNB  P1.3,LABEL1
JNB  ACC.3,LABEL2
```

will cause program execution to continue at the instruction at label LABEL2.

**Bytes:** 3**Cycles:** 2**Encoding:**

|   |   |   |   |   |   |   |   |             |              |
|---|---|---|---|---|---|---|---|-------------|--------------|
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | bit address | rel. address |
|---|---|---|---|---|---|---|---|-------------|--------------|

**Operation:**

```
JNB
(PC) ← (PC) + 3
IF (bit) = 0
  THEN (PC) ← (PC) + rel.
```

**JNC     rel****Function:** Jump if Carry not set**Description:** If the carry flag is a zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice to point to the next instruction. The carry flag is not modified.**Example:** The carry flag is set. The instruction sequence,

```
JNC LABEL1
CPL C
JNC LABEL2
```

will clear the carry and cause program execution to continue at the instruction identified by the label LABEL2.



|                   |   |              |
|-------------------|---|--------------|
| <b>Bytes:</b>     | 2   |              |
| <b>Cycles:</b>    | 2   |              |
| <b>Encoding:</b>  | 0 1 0 1 0 0 0 0   | rel. address |
| <b>Operation:</b> | JNC<br>$(PC) \leftarrow (PC) + 2$<br>IF (C) = 0<br>THEN $(PC) \leftarrow (PC) + \text{rel}$ |              |

# **JNZ rel**

|                     |  |
|---------------------|--|
| <b>Function:</b>    | Jump if accumulator Not Zero   |
| <b>Description:</b> | If any bit of the accumulator is a one, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The accumulator is not modified. No flags are affected. |
| <b>Example:</b>     | The accumulator originally holds 00H. The instruction sequence,<br><pre> JNZ LABEL1 INC A JNZ LABEL2                     </pre> will set the accumulator to 01H and continue at label LABEL2.  |
| <b>Bytes:</b>       | 2  |
| <b>Cycles:</b>      | 2  |

|                   |  |              |
|-------------------|--|--------------|
| <b>Encoding:</b>  | 0 1 1 1 0 0 0 0  | rel. address |
| <b>Operation:</b> | JNZ<br>$(PC) \leftarrow (PC) + 2$<br>IF (A) $\neq 0$<br>THEN $(PC) \leftarrow (PC) + \text{rel}$ |              |

**JZ**      **rel**

**Function:** Jump if Accumulator Zero  
**Description:** If all bits of the accumulator are zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The accumulator is not modified. No flags are affected.

**Example:** The accumulator originally contains 01H. The instruction sequence,

```
JZ LABEL1
DEC A
JZ LABEL2
```

will change the accumulator to 00H and cause program execution to continue at the instruction identified by the label LABEL2.

**Bytes:** 2  
**Cycles:** 2

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

rel. address

**Operation:** JZ  
 $(PC) \leftarrow (PC) + 2$   
 IF (A) = 0  
     THEN  $(PC) \leftarrow (PC) + \text{rel}$

**LCALL**      **addr16**

**Function:** Long Call  
**Description:** LCALL calls a subroutine located at the indicated address. The instruction adds three to the program counter to generate the address of the next instruction and then pushes the 16-bit result onto the stack (low byte first), incrementing the stack pointer by two. The high-order and low-order bytes of the PC are then loaded, respectively, with the second and third bytes of the LCALL instruction. Program execution continues with the instruction at this address. The subroutine may therefore begin anywhere in the full 64K-byte program memory address space. No flags are affected.

**Example:** Initially the stack pointer equals 07H. The label "SUBRTN" is assigned to program memory location 1234H. After executing the instruction,

LCALL SUBRTN

at location 0123H, the stack pointer will contain 09H, internal RAM locations 08H and 09H will contain 26H and 01H, and the PC will contain 1235H.

**Bytes:** 3

**Cycles:** 2

**Encoding:**

|   |   |   |   |   |   |   |   |                |               |
|---|---|---|---|---|---|---|---|----------------|---------------|
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | addr15 - addr8 | addr7 - addr0 |
|---|---|---|---|---|---|---|---|----------------|---------------|

**Operation:**

LCALL

$(PC) \leftarrow (PC) + 3$

$(SP) \leftarrow (SP) + 1$

$((SP)) \leftarrow (PC7-0)$

$(SP) \leftarrow (SP) + 1$

$((SP)) \leftarrow (PC15-8)$

$(PC) \leftarrow \text{addr15-0}$

**LJMP addr16**

**Function:** Long Jump

**Description:** LJMP causes an unconditional branch to the indicated address, by loading the high-order and low-order bytes of the PC (respectively) with the second and third instruction bytes. The destination may therefore be anywhere in the full 64K program memory address space. No flags are affected.

**Example:** The label "JMPADR" is assigned to the instruction at program memory location 1234H. The instruction,

LJMP JMPADR

at location 0123H will load the program counter with 1234H.

**Bytes:** 3

**Cycles:** 2

**Encoding:**

|   |   |   |   |   |   |   |   |                |               |
|---|---|---|---|---|---|---|---|----------------|---------------|
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | addr15 - addr8 | addr7 - addr0 |
|---|---|---|---|---|---|---|---|----------------|---------------|

**Operation:**

LJMP

$(PC) \leftarrow \text{addr15-0}$

**MOV** <dest-byte>, <src-byte>**Function:** Move byte variable**Description:** The byte variable indicated by the second operand is copied into the location specified by the first operand. The source byte is not affected. No other register or flag is affected.

This is by far the most flexible operation. Fifteen combinations of source and destination addressing modes are allowed.

**Example:** Internal RAM location 30H holds 40H. The value of RAM location 40H is 10H. The data present at input port 1 is 11001010B (0CAH).

```
MOV R0,#30H      ;R0<= 30H
MOV A,@R0        ;A <= 40H
MOV R1,A         ;R1 <= 40H
MOV R,@R1        ;B <= 10H
MOV @R1,P1       ;RAM (40H) <= 0CAH
MOV P2, P1       ;P2 #0CAH
```

leaves the value 30H in register 0, 40H in both the accumulator and register 1, 10H in register B, and 0CAH (11001010B) both in RAM location 40H and output on port 2.

**MOV A,Rn****Bytes:** 1**Cycles:** 1**Encoding:** 1 1 1 0 1 r r r**Operation:** MOV  
(A) ← (Rn)**\*MOV A,direct****Bytes:** 2**Cycles:** 1**Encoding:** 1 1 1 0 0 1 0 1      direct address**Operation:** MOV  
(A) ← (direct)**MOV A,ACC is not a valid instruction.**

**MOV A,@Ri**

Bytes: 1

Cycles: 1

Encoding:

1 1 1 0 0 1 1 1 i

Operation:

MOV

(A) ← ((Ri))

**MOV A,#data**

Bytes: 2

Cycles: 1

Encoding:

0 1 1 1 0 1 0 0 0 immediate data

Operation:

MOV

(A) ← #data

**MOV Rn,A**

Bytes: 1

Cycles: 1

Encoding:

1 1 1 1 1 1 r r r

Operation:

MOV

(Rn) ← (A)

**MOV Rn,direct**

Bytes: 2

Cycles: 2

Encoding:

1 0 1 0 1 r r r

direct addr.

Operation:

MOV

(Rn) ← (direct)

**MOV Rn,#data**

Bytes: 2

Cycles: 1

Encoding:

0 1 1 1 1 r r r

immediate data

Operation:

MOV

(Rn) ← #data



**MOV direct,A**
**Bytes:** 2

**Cycles:** 1

**Encoding:**

1 1 1 1 0 1 0 1

direct address

**Operation:**

MOV

(direct) ← (A)

**MOV direct,Rn**
**Bytes:** 2

**Cycles:** 2

**Encoding:**

1 0 0 0 1 r r r r

direct address

**Operation:**

MOV

(direct) ← (Rn)

**MOV direct,direct**
**Bytes:** 3

**Cycles:** 2

**Encoding:**

1 0 0 0 0 1 0 1

dir. addr. (src)

dir. addr. (dest)

**Operation:**

MOV

(direct) ← (direct)

**MOV direct,@Ri**
**Bytes:** 2

**Cycles:** 2

**Encoding:**

1 0 0 0 0 1 1 i

direct addr.

**Operation:**

MOV

(direct) ← ((Ri))

**MOV direct,#data**
**Bytes:** 3

**Cycles:** 2

**Encoding:**

0 1 1 1 0 1 0 1

direct address

immediate data

**Operation:**

MOV

(direct) ← #data

**MOV @Ri,A**

**Bytes:** 1

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 | 1 | 1 | i |
|---|---|---|---|---|---|---|---|

**Operation:** MOV  
((Ri)) ← (A)

**MOV @Ri,direct**

**Bytes:** 2

**Cycles:** 2

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | i |
|---|---|---|---|---|---|---|---|

|              |
|--------------|
| direct addr. |
|--------------|

**Operation:** MOV  
((Ri)) ← (direct)

**MOV @Ri,#data**

**Bytes:** 2

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 0 | 1 | 1 | i |
|---|---|---|---|---|---|---|---|

|                |
|----------------|
| immediate data |
|----------------|

**Operation:** MOV  
((RI)) ← #data

**MOV <dest-bit>,<src-bit>**

**Function:** Move bit data

**Description:** The Boolean variable indicated by the second operand is copied into the location specified by the first operand. One of the operands must be the carry flag; the other may be any directly addressable bit. No other register or flag is affected.

**Example:** The carry flag is originally set. The data present at input port 3 is 11000101B. The data previously written to output port 1 is 35H (00110101B).

MOV P1.3,C

MOV C,P3.3

MOV P1.2,C

will leave the carry cleared and change port 1 to 39H (00111001B).

**MOV C,bit**

**Bytes:** 2

**Cycles:** 1

**Encoding:**

1 0 1 0 0 0 1 0

bit address

**Operation:**

MOV

(C) ← (bit)

**MOV bit,C**

**Bytes:** 2

**Cycles:** 2

**Encoding:**

1 0 0 1 0 0 1 0

bit address

**Operation:**

MOV

(bit) ← (C)

**MOV DPTR,#data16**

**Function:**

Load Data Pointer with a 16-bit constant

**Description:**

The data pointer is loaded with the 16-bit constant indicated. The 16-bit constant is loaded into the second and third bytes of the instruction. The second byte (DPH) is the high-order byte, while the third byte (DPL) holds the low-order byte. No flags are affected.

This is the only instruction which moves 16 bits of data at once.

**Example:**

The instruction,

MOV DPTR,#1234H

will load the value 1234H into the data pointer: DPH will hold 12H and DPL will hold 34H.

**Bytes:** 3

**Cycles:** 2

**Encoding:**

1 0 0 1 0 0 0 0

immed. data15 - 8

immed. data7 - 0

**Operation:**

MOV

(DPTR) ← #data15-0

DPH ← DPL ← #data15-8 ← #data7-0

**MOVC A,@A + <base-reg>**

**Function:** Move Code byte

**Description:** The MOVC instructions load the accumulator with a code byte, or constant from program memory. The address of the byte fetched is the sum of the original unsigned eight-bit accumulator contents and the contents of a sixteen-bit base register, which may be either the data pointer or the PC. In the latter case, the PC is incremented to the address of the following instruction before being added with the accumulator; otherwise the base register is not altered. Sixteen-bit addition is performed so a carry-out from the low-order eight bits may propagate through higher-order bits. No flags are affected.

**Example:** A value between 0 and 3 is in the accumulator. The following instructions will translate the value in the accumulator to one of four values defined by the DB (define byte) directive.

```
REL_PC: INC      A
        MOVC     A,@A+PC
        RET
        DB       66H
        DB       77H
        DB       88H
        DB       99H
```

If the subroutine is called with the accumulator equal to 01H, it will return with 77H in the accumulator. The INC A before the MOVC instruction is needed to "get around" the RET instruction above the table. If several bytes of code separated the MOVC from the table, the corresponding number would be added to the accumulator instead.

**MOVC A,@A + DPTR**

**Bytes:** 1

**Cycles:** 2

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|

**Operation:** MOVC  
(A) ← ((A) + (DPTR))

|                             |   |
|-----------------------------|---|
| MCS-51 PROGRAMMING          |   |
| MOVX A,@A+PC                | MOVX A,@A+PC  |
| Bytes:                      | 1   |
| Cycles:                     | 2   |
| Encoding:                   | 1 0 0 0 0 0 0 1 1   |
| Operation:                  | MOVX<br>$(PC) \leftarrow (PC) + 1$<br>$(A) \leftarrow ((A) + (PC))$ |
| Example:                    | Translate the value in the accumulator to                           |
| MOVX <dest-byte>,<src-byte> |   |

**Function:** Move External  
**Description:** The MOVX instructions transfer data between the accumulator and a byte of external data memory, hence the "X" appended to MOV. There are two types of instructions, differing in whether they provide an eight-bit or sixteen-bit indirect address to the external data RAM.

In the first type, the contents of R0 or R1 in the current register bank provide an eight-bit address multiplexed with data on P0. Eight bits are sufficient for external I/O expansion decoding or for a relatively small RAM array. For somewhat larger arrays, any output port pins can be used to output higher-order address bits. These pins would be controlled by an output instruction preceding the MOVX.

In the second type of MOVX instruction, the data pointer generates a sixteen-bit address. P2 outputs the high-order eight address bits (the contents of DPH) while P0 multiplexes the low-order eight bits (DPL) with data. The P2 Special Function Register retains its previous contents while the P2 output buffers are emitting the contents of DPH. This form is faster and more efficient when accessing very large data arrays (up to 64K bytes), since no additional instructions are needed to set up the output ports.

It is possible in some situations to mix the two MOVX types. A large RAM array with its high-order address lines driven by P2 can be addressed via the data pointer, or with code to output high-order address bits to P2 followed by a MOVX instruction using R0 or R1.



**Example:** An external 256 byte RAM using multiplexed address/data lines (e.g., an Intel® 8155 RAM/I/O/Timer) is connected to the 8051 Port 0. Port 3 provides control lines for the external RAM. Ports 1 and 2 are used for normal I/O. Registers 0 and 1 contain 12H and 34H. Location 34H of the external RAM holds the value 56H. The instruction sequence,

```
MOVX A,@R1
MOVX @R0,A
```

copies the value 56H into both the accumulator and external RAM location 12H.

**MOVX A,@Ri**

Bytes: 1

Cycles: 2

Encoding:

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 0 | 1 | i |
|---|---|---|---|---|---|---|---|

Operation:

MOVX

(A) ← ((Ri))

**MOVX A,@DPTR**

Bytes: 1

Cycles: 2

Encoding:

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

Operation:

MOVX

(A) ← ((DPTR))

**MOVX @Ri,A**

Bytes: 1

Cycles: 2

Encoding:

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 | 0 | 1 | i |
|---|---|---|---|---|---|---|---|

Operation:

MOVX

((Ri)) ← (A)

**MOVX @DPTR,A**

Bytes: 1

Cycles: 2

Encoding:

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

Operation:

MOVX

(DPTR) ← (A)

**NOP****Function:** No Operation**Description:** Execution continues at the following instruction. Other than the PC, no registers or flags are affected.**Example:** It is desired to produce a low-going output pulse on bit 7 of port 2 lasting exactly 5 cycles. A simple SETB/CLR sequence would generate a one-cycle pulse, so four additional cycles must be inserted. This may be done (assuming no interrupts are enabled) with the instruction sequence,

CLR P2.7  
 NOP  
 NOP  
 NOP  
 NOP  
 SETB P2.7

**Bytes:** 1  
**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

**Operation:** NOP  
 (PC) ← (PC) + 1

MUL AB

**Function:** Multiply

**Description:** MUL AB multiplies the unsigned eight-bit integers in the accumulator and register B. The low-order byte of the sixteen-bit product is left in the accumulator, and the high-order byte in B. If the product is greater than 255 (0FFH) the overflow flag is set; otherwise it is cleared. The carry flag is always cleared.

**Example:** Originally the accumulator holds the value 80 (50H). Register B holds the value 160 (0A0H). The instruction,

MUL AB

will give the product 12,800 (3200H), so B is changed to 32H (00110010B) and the accumulator is cleared. The overflow flag is set, carry is cleared.

**Bytes:** 1  
**Cycles:** 4

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|

**Operation:** MUL  
 $(A)_{7-0} \leftarrow (A) \times (B)$   
 $(B)_{15-8}$

**ORL**    <dest-byte> <src-byte>**Function:** Logical-OR for byte variables**Description:** ORL performs the bitwise logical-OR operation between the indicated variables, storing the results in the destination byte. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the accumulator or immediate data.

*Note:* When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

**Example:** If the accumulator holds 0C3H (11000011B) and R0 holds 55H (01010101B) then the instruction,

```
ORL    A,R0
```

will leave the accumulator holding the value 0D7H (11010111B).

When the destination is a directly addressed byte, the instruction can set combinations of bits in any RAM location or hardware register. The pattern of bits to be set is determined by a mask byte, which may be either a constant data value in the instruction or a variable computed in the accumulator at run-time. The instruction,

```
ORL    P1,#00110010B
```

will set bits 5, 4, and 1 of output port 1.

**ORL A,Rn****Bytes:** 1**Cycles:** 1**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 | r | r | r |
|---|---|---|---|---|---|---|---|

**Operation:**

ORL

 $(A) \leftarrow (A) \vee (Rn)$

**ORL A,direct**

Bytes: 2

Cycles: 1

Encoding: 

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|

 direct address

Operation: ORL  
(A) ← (A) ∨ (direct)

**ORL A,@Ri**

Bytes: 1

Cycles: 1

Encoding: 

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | i |
|---|---|---|---|---|---|---|---|

Operation: ORL  
(A) ← (A) ∨ ((Ri))

**ORL A,#data**

Bytes: 2

Cycles: 1

Encoding: 

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|

 immediate data

Operation: ORL  
(A) ← (A) ∨ #data

**ORL direct,A**

Bytes: 2

Cycles: 1

Encoding: 

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|---|

 direct address

Operation: ORL  
(direct) ← (direct) ∨ (A)

**ORL direct,#data**

Bytes: 3

Cycles: 2

Encoding: 

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|

 direct addr. immediate data

Operation: ORL  
(direct) ← (direct) ∨ #data



**ORL C, <src-bit>****Function:** Logical-OR for bit variables**Description:** Set the carry flag if the Boolean value is a logical 1; leave the carry in its current state otherwise. A slash ("/") preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, but the source bit itself is not affected. No other flags are affected.**Example:** Set the carry flag if and only if P1.0 = 1, ACC.7 = 1, or OV = 0:

```

MOV C,P1.0           ;LOAD CARRY WITH INPUT PIN P10
ORL C,ACC.7           ;OR CARRY WITH THE ACC. BIT 7
ORL C,/OV             ;OR CARRY WITH THE INVERSE OF OV

```

**ORL C,bit****Bytes:** 2**Cycles:** 2**Encoding:**

0 1 1 1 0 0 1 0

bit address

**Operation:**

ORL  
 $(C) \leftarrow (C) \vee (\text{bit})$

**ORL C,bit****Bytes:** 2**Cycles:** 2**Encoding:**

1 0 1 0 0 0 0 0

bit address

**Operation:**

ORL  
 $(C) \leftarrow (C) \vee (\overline{\text{bit}})$

**POP direct****Function:** Pop from stack.**Description:** The contents of the internal RAM location addressed by the stack pointer is read, and the stack pointer is decremented by one. The value read is then transferred to the directly addressed byte indicated. No flags are affected.

**Example:** The stack pointer originally contains the value 32H, and internal RAM locations 30H through 32H contain the values 20H, 23H, and 01H, respectively. The instruction sequence,

```
POP    DPH
POP    DPL
```

will leave the stack pointer equal to the value 30H and the data pointer set to 0123H. At this point the instruction,

```
POP    SP
```

will leave the stack pointer set to 20H. Note that in this special case the stack pointer was decremented to 2FH before being loaded with the value popped (20H).

**Bytes:** 2

**Cycles:** 2

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

direct address

**Operation:**

POP

(direct) ← ((SP))

(SP) ← (SP) - 1

**PUSH      direct**

**Function:** Push onto stack

**Description:** The stack pointer is incremented by one. The contents of the indicated variable is then copied into the internal RAM location addressed by the stack pointer. Otherwise no flags are affected.

**Example:** On entering an interrupt routine the stack pointer contains 09H. The data pointer holds the value 0123H. The instruction sequence,

```
PUSH    DPL
PUSH    DPH
```

will leave the stack pointer set to 0BH and store 23H and 01H in internal RAM locations 0AH and 0BH, respectively.

**Bytes:** 2

**Cycles:** 2

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

direct address

**Operation:**

PUSH

(SP) ← (SP) + 1

((SP)) ← (direct)

## RET

|                     |  |   |   |   |   |   |   |   |   |
|---------------------|--|---|---|---|---|---|---|---|---|
| <b>Function:</b>    | Return from subroutine   |   |   |   |   |   |   |   |   |
| <b>Description:</b> | RET pops the high- and low-order bytes of the PC successively from the stack, decrementing the stack pointer by two. Program execution continues at the resulting address, generally the instruction immediately following an ACALL or LCALL. No flags are affected. |   |   |   |   |   |   |   |   |
| <b>Example:</b>     | The stack pointer originally contains the value 0BH. Internal RAM locations 0AH and 0BH contain the values 23H and 01H, respectively. The instruction, RET will leave the stack pointer equal to the value 09H. Program execution will continue at location 0123H.   |   |   |   |   |   |   |   |   |
| <b>Bytes:</b>       | 1  |   |   |   |   |   |   |   |   |
| <b>Cycles:</b>      | 2  |   |   |   |   |   |   |   |   |
| <b>Encoding:</b>    | <table><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr></table>   | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0                   | 0  | 1 | 0 | 0 | 0 | 1 | 0 |   |   |
| <b>Operation:</b>   | RET<br>$(PC_{15-8}) \leftarrow ((SP))$<br>$(SP) \leftarrow (SP) - 1$<br>$(PC_{7-0}) \leftarrow ((SP))$<br>$(SP) \leftarrow (SP) - 1$   |   |   |   |   |   |   |   |   |

## RETI

|                     |   |
|---------------------|---|
| <b>Function:</b>    | Return from interrupt   |
| <b>Description:</b> | RETI pops the high- and low-order bytes of the PC successively from the stack, and restores the interrupt logic to accept additional interrupts at the same priority level as the one just processed. The stack pointer is left decremented by two. No other registers are affected; the PSW is <i>not</i> automatically restored to its pre-interrupt status. Program execution continues at the resulting address, which is generally the instruction immediately after the point at which the interrupt request was detected. If a lower- or same-level interrupt had been pending when the RETI instruction is executed, that one instruction will be executed before the pending interrupt is processed. |

**Example:** The stack pointer originally contains the value 0BH. An interrupt was detected during the instruction ending at location 0122H. Internal RAM locations 0AH and 0BH contain the values 23H and 01H, respectively. The instruction,

RETI

will leave the stack pointer equal to 09H and return program execution to location 0123H.

**Bytes:** 1

**Cycles:** 2

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|---|

**Operation:**

RETI

$(PC_{15-8}) \leftarrow ((SP))$

$(SP) \leftarrow (SP) - 1$

$(PC_{7-0}) \leftarrow ((SP))$

$(SP) \leftarrow (SP) - 1$

**RL    A****Function:** Rotate accumulator Left**Description:** The eight bits in the accumulator are rotated one bit to the left. Bit 7 is rotated into the bit 0 position. No flags are affected.**Example:** The accumulator holds the value 0C5H (11000101B). The instruction,

RL    A

leaves the accumulator holding the value 8BH (10001011B) with the carry unaffected.

**Bytes:**

1

**Cycles:**

1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|

**Operation:**

RL

 $(A_{n+1}) \leftarrow (A_n) \quad n = 0-6$  $(A_0) \leftarrow (A_7)$ **RLC    A****Function:** Rotate accumulator Left through the Carry flag**Description:** The eight bits in the accumulator and the carry flag are together rotated one bit to the left. Bit 7 moves into the carry flag; the original state of the carry flag moves into the bit 0 position. No other flags are affected.**Example:** The accumulator holds the value 0C5H (11000101B), and the carry is zero. The instruction,

RLC    A

leaves the accumulator holding the value 8BH (10001010B) with the carry set.

**Bytes:**

1

**Cycles:**

1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|

**Operation:**

RLC

 $(A_{n+1}) \leftarrow (A_n) \quad n = 0-6$  $(A_0) \leftarrow (C)$  $(C) \leftarrow (A_7)$



**RR A**

**Function:** Rotate accumulator Right

**Description:** The eight bits in the accumulator are rotated one bit to the right. Bit 0 is rotated into the bit 7 position. No flags are affected.

**Example:** The accumulator holds the value 0C5H (11000101B). The instruction,

RR A

leaves the accumulator holding the value 0E2H (11100010B) with the carry unaffected.

**Bytes:** 1

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|

**Operation:**

RR

$(A_n) \leftarrow (A_{n+1}) \quad n=0-6$

$(A_7) \leftarrow (A_0)$

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|

**RRC A**

**Function:** Rotate accumulator Right through Carry flag

**Description:** The eight bits in the accumulator and the carry flag are together rotated one bit to the right. Bit 0 moves into the carry flag; the original value of the carry flag moves into the bit 7 position. No other flags are affected.

**Example:** The accumulator holds the value 0C5H (11000101B), the carry is zero. The instruction,

RRC A

leaves the accumulator holding the value 62 (01100010B) with the carry set.

**Bytes:** 1

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|

**Operation:**

RRC

$(A_n) \leftarrow (A_{n+1}) \quad n=0-6$

$(A_7) \leftarrow (C)$

$(C) \leftarrow (A_0)$

# SETB <bit>

**Function:** Set Bit

**Description:** SETB sets the indicated bit to one. SETB can operate on the carry flag or any directly addressable bit. No other flags are affected.

**Example:** The carry flag is cleared. Output port 1 has been written with the value 34H (00110100B). The instructions,

```
SETB C
SETB P1.0
```

will leave the carry flag set to 1 and change the data output on port 1 to 35H (00110101B).

## SETB C

**Bytes:** 1

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|

**Operation:**

SETB  
(C) ← 1

## SETB bit

**Bytes:** 2

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|---|

bit address

**Operation:**

SETB  
(bit) ← 1

# SJMP rel

**Function:** Short Jump

**Description:** Program control branches unconditionally to the address indicated. The branch destination is computed by adding the signed displacement in the second instruction byte to the PC, after incrementing the PC twice. Therefore, the range of destinations allowed is from 128 bytes preceding this instruction to 127 bytes following it.

**Example:** The label "RELADR" is assigned to an instruction at program memory location 0123H. The instruction,

SJMP RELADR

will assemble into location 0100H. After the instruction is executed, the PC will contain the value 0123H.

(Note: Under the above conditions the instruction following SJMP will be at 102H. Therefore, the displacement byte of the instruction will be the relative offset (0123H-0102H) = 21H. Put another way, an SJMP with a displacement of 0FEH would be a one-instruction infinite loop.)

**Bytes:** 2

**Cycles:** 2

**Encoding:** 1 0 0 0 0 0 0 0 rel. address

**Operation:** SJMP  
 $(PC) \leftarrow (PC) + 2$   
 $(PC) \leftarrow (PC) + \text{rel}$

**SUBB A, <src-byte>****Function:** Subtract with borrow

**Description:** SUBB subtracts the indicated variable and the carry flag together from the accumulator, leaving the result in the accumulator. SUBB sets the carry (borrow) flag if a borrow is needed for bit 7, and clears C otherwise. (If C was set *before* executing a SUBB instruction, this indicates that a borrow was needed for the previous step in a multiple precision subtraction, so the carry is subtracted from the accumulator along with the source operand.) AC is set if a borrow is needed for bit 3, and cleared otherwise. OV is set if a borrow is needed into bit 6, but not into bit 7, or into bit 7, but not bit 6.

When subtracting signed integers OV indicates a negative number produced when a negative value is subtracted from a positive value, or a positive result when a positive number is subtracted from a negative number.

The source operand allows four addressing modes: register, direct, register-indirect, or immediate.

**Example:** The accumulator holds 0C9H (11001001B), register 2 holds 54H (01010100B), and the carry flag is set. The instruction,

SUBB A,R2

will leave the value 74H (01110100B) in the accumulator, with the carry flag and AC cleared but OV set.

Notice that 0C9H minus 54H is 75H. The difference between this and the above result is due to the carry (borrow) flag being set before the operation. If the state of the carry is not known before starting a single or multiple-precision subtraction, it should be explicitly cleared by a CLR C instruction.

**SUBB A,Rn****Bytes:** 1**Cycles:** 1**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 | r | r | r |
|---|---|---|---|---|---|---|---|

**Operation:**

SUBB

 $(A) \leftarrow (A) - (C) - (Rn)$

**SUBB A,direct****Bytes:** 2**Cycles:** 1**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|

 direct address**Operation:** SUBB $(A) \leftarrow (A) - (C) - (\text{direct})$ **SUBB A,@Ri****Bytes:** 1**Cycles:** 1**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|

**Operation:** SUBB $(A) \leftarrow (A) - (C) - ((Ri))$ **SUBB A,#data****Bytes:** 2**Cycles:** 1**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|

 immediate data**Operation:** SUBB $(A) \leftarrow (A) - (C) - \#data$ **SWAP A****Function:** Swap nibbles within the Accumulator**Description:** SWAP A interchanges the low- and high-order nibbles (four-bit fields) of the accumulator (bits 3-0 and bits 7-4). The operation can also be thought of as a four-bit rotate instruction. No flags are affected.**Example:** The accumulator holds the value 0C5H (11000101B). The instruction,

SWAP A

leaves the accumulator holding the value 5CH (01011100B).

**Bytes:** 1**Cycles:** 1**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|

**Operation:** SWAP $(A_{3-0}) \leftrightarrow (A_{7-4}), (A_{7-4}) \leftarrow (A_{3-0})$



# XCH A,<byte>

**Function:** Exchange Accumulator with byte variable

**Description:** XCH loads the accumulator with the contents of the indicated variable, at the same time writing the original accumulator contents to the indicated variable. The source/destination operand can use register, direct, or register-indirect addressing.

**Example:** R0 contains the address 20H. The accumulator holds the value 3FH (00111111B). Internal RAM location 20H holds the value 75H (01110101B). The instruction,

XCH A,@R0

will leave RAM location 20H holding the values 3FH (00111111B) and 75H (01110101B) in the accumulator.

## XCH A,Rn

**Bytes:** 1

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 1 | r | r | r |
|---|---|---|---|---|---|---|---|

**Operation:**

XCH  
(A)  $\longleftrightarrow$  (Rn)

## XCH A,direct

**Bytes:** 2

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|

direct address

**Operation:**

XCH  
(A)  $\longleftrightarrow$  (direct)

## XCH A,@Ri

**Bytes:** 1

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | i |
|---|---|---|---|---|---|---|---|

**Operation:**

XCH  
(A)  $\longleftrightarrow$  ((Ri))

XCHD      A,@Ri

**Function:** Exchange Digit**Description:** XCHD exchanges the low-order nibble of the accumulator (bits 3-0), generally representing a hexadecimal or BCD digit), with that of the internal RAM location indirectly addressed by the specified register. The high-order nibbles (bits 7-4) of each register are not affected. No flags are affected.**Example:** R0 contains the address 20H. The accumulator holds the value 36H (00110110B). Internal RAM location 20H holds the value 75H (01101010B). The instruction,

XCHD A,@R0

will leave RAM location 20H holding the value 76H (0110110B) and 35H (00110101B) in the accumulator.

**Bytes:** 1**Cycles:** 1**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 0 | 1 | 1 | i |
|---|---|---|---|---|---|---|---|

**Operation:**

XCHD

$$(A3-0) \longleftrightarrow ((Ri3-0))$$

**XRL**    <dest-byte>, <src-byte>**Function:** Logical Exclusive-OR for byte variables**Description:** XRL performs the bitwise logical Exclusive-OR operation between the indicated variables, storing the results in the destination. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the accumulator or immediate data.

(Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.)

**Example:** If the accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B) then the instruction,

XRL    A,R0

will leave the accumulator holding the value 69H (01101001B).

When the destination is a directly addressed byte, this instruction can complement combinations of bits in any RAM location or hardware register. The pattern of bits to be complemented is then determined by a mask byte, either a constant contained in the instruction or a variable computed in the accumulator at run-time. The instruction,

XRL    P1,#00110001B

will complement bits 5, 4, and 0 of output port 1.

**XRL**    A,Rn**Bytes:**

1

**Cycles:**

1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 1 | r | r | r |
|---|---|---|---|---|---|---|---|

**Operation:**

XRL

 $(A) \leftarrow (A) \vee (Rn)$ **XRL**    A,direct**Bytes:**

2

**Cycles:**

1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|

|                |
|----------------|
| direct address |
|----------------|

**Operation:**

XRL

 $(A) \leftarrow (A) \vee (\text{direct})$

**XRL A,@Ri****Bytes:** 1**Cycles:** 1**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | i |
|---|---|---|---|---|---|---|---|

**Operation:**

XRL

 $(A) \leftarrow (A) \vee ((Ri))$ **XRL A,#data****Bytes:** 2**Cycles:** 1**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|

immediate data

**Operation:**

XRL

 $(A) \leftarrow (A) \vee \#data$ **XRL direct,A****Bytes:** 2**Cycles:** 1**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|---|

direct address

**Operation:**

XRL

 $(direct) \leftarrow (direct) \vee (A)$ **XRL direct,#data****Bytes:** 3**Cycles:** 2**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|

direct address

immediate data

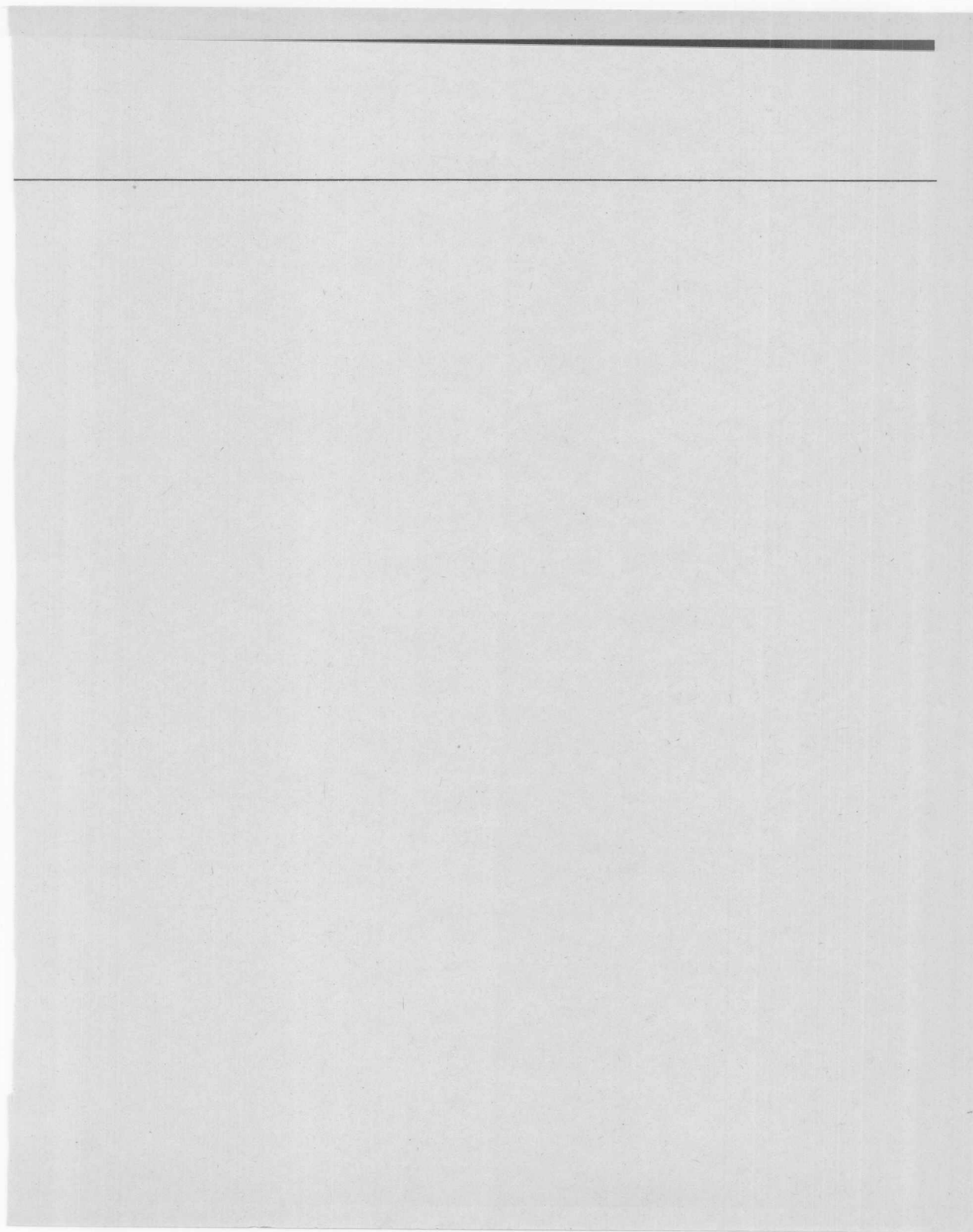
**Operation:**

XRL

 $(direct) \leftarrow (direct) \vee \#data$







MCS®-51 Data Sheets

9

## MCS<sup>®</sup>-51 8-BIT CONTROL-ORIENTED MICROCOMPUTERS

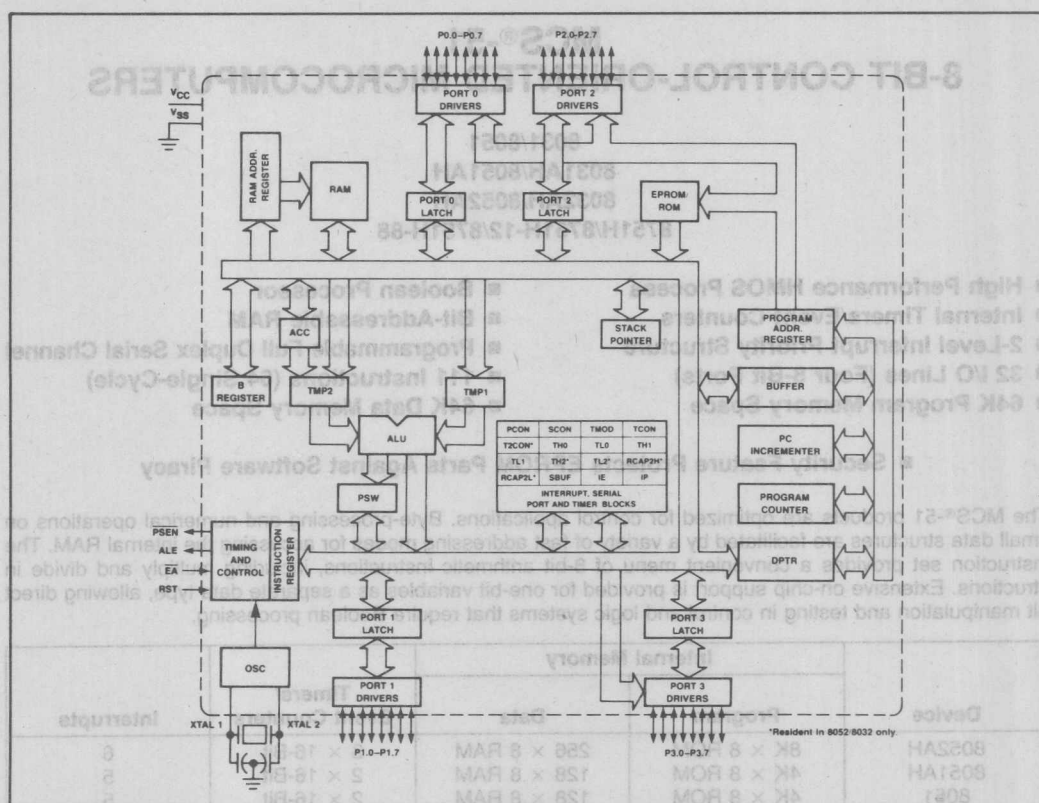
8031/8051  
8031AH/8051AH  
8032AH/8052AH  
8751H/8751H-12/8751H-88

- High Performance HMOS Process
- Internal Timers/Event Counters
- 2-Level Interrupt Priority Structure
- 32 I/O Lines (Four 8-Bit Ports)
- 64K Program Memory Space
- Boolean Processor
- Bit-Addressable RAM
- Programmable Full Duplex Serial Channel
- 111 Instructions (64 Single-Cycle)
- 64K Data Memory Space
- Security Feature Protects EPROM Parts Against Software Piracy

The MCS<sup>®</sup>-51 products are optimized for control applications. Byte-processing and numerical operations on small data structures are facilitated by a variety of fast addressing modes for accessing the internal RAM. The instruction set provides a convenient menu of 8-bit arithmetic instructions, including multiply and divide instructions. Extensive on-chip support is provided for one-bit variables as a separate data type, allowing direct bit manipulation and testing in control and logic systems that require Boolean processing.

| Device   | Internal Memory |             | Timers/<br>Event Counters | Interrupts |
|----------|-----------------|-------------|---------------------------|------------|
|          | Program         | Data        |                           |            |
| 8052AH   | 8K × 8 ROM      | 256 × 8 RAM | 3 × 16-Bit                | 6          |
| 8051AH   | 4K × 8 ROM      | 128 × 8 RAM | 2 × 16-Bit                | 5          |
| 8051     | 4K × 8 ROM      | 128 × 8 RAM | 2 × 16-Bit                | 5          |
| 8032AH   | none            | 256 × 8 RAM | 3 × 16-Bit                | 6          |
| 8031AH   | none            | 128 × 8 RAM | 2 × 16-Bit                | 5          |
| 8031     | none            | 128 × 8 RAM | 2 × 16-Bit                | 5          |
| 8751H    | 4K × 8 EPROM    | 128 × 8 RAM | 2 × 16-Bit                | 5          |
| 8751H-12 | 4K × 8 EPROM    | 128 × 8 RAM | 2 × 16-Bit                | 5          |
| 8751H-88 | 4K × 8 EPROM    | 128 × 8 RAM | 2 × 16-Bit                | 5          |

The 8751H is an EPROM version of the 8051AH; that is, the on-chip Program Memory can be electrically programmed, and can be erased by exposure to ultraviolet light. It is fully compatible with its predecessor, the 8751-8, but incorporates two new features: a Program Memory Security bit that can be used to protect the EPROM against unauthorized read-out, and a programmable baud rate modification bit (SMOD). SMOD is not present in the 8751H-12 or the 8751H-88. The 8751H-88 also only operates up to 8 MHz.



**Figure 1. MCS<sup>®</sup>-51 Block Diagram**

## PIN DESCRIPTIONS

## VCC

Supply voltage.

## VSS

Circuit ground.

## Port 0

Port 0 is an 8-bit open drain bidirectional I/O port. As an output port each pin can sink 8 LS TTL inputs. Port 0 pins that have 1s written to them float, and in that state can be used as high-impedance inputs.

Port 0 is also the multiplexed low-order address and data bus during accesses to external Program and Data Memory. In this application it uses strong internal pullups when emitting 1s, and can source and sink 8 LS TTL inputs.

Port 0 also receives the code bytes during programming of the EPROM parts, and outputs the code bytes during program verification of the ROM and EPROM parts. External pullups are required during program verification.

## Port 1

Port 1 is an 8-bit bidirectional I/O port with internal pullups. The Port 1 output buffers can sink/source 4 LS TTL inputs. Port 1 pins that have 1s written to them are pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 1 pins that are externally being pulled low will source current (IIL, on the data sheet) because of the internal pullups.

Port 1 also receives the low-order address bytes during programming of the EPROM parts and during program verification of the ROM and EPROM parts.

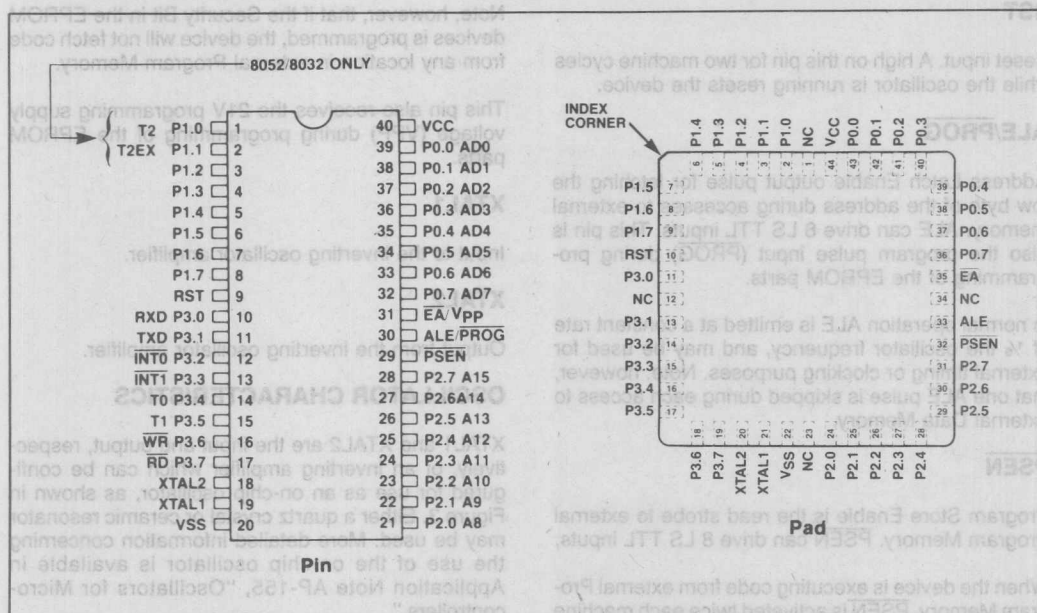


Figure 2. MCS-51 Connections

In the 8032AH and 8052AH, Port 1 pins P1.0 and P1.1 also serve the T2 and T2EX functions, respectively.

## Port 2

Port 2 is an 8-bit bidirectional I/O port with internal pullups. The Port 2 output buffers can sink/source 4 LS TTL inputs. Port 2 pins that have 1s written to them are pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 2 pins that are externally being pulled low will source current (IIL, on the data sheet) because of the internal pullups.

Port 2 emits the high-order address byte during fetches from external Program Memory and during accesses to external Data Memory that use 16-bit addresses (MOVX @DPTR). In this application it uses strong internal pullups when emitting 1s. During accesses to external Data Memory that use 8-bit addresses (MOVX @Ri), Port 2 emits the contents of the P2 Special Function Register.

Port 2 also receives the high-order address bits during programming of the EPROM parts and during program verification of the ROM and EPROM parts.

## Port 3

Port 3 is an 8-bit bidirectional I/O port with internal pullups. The Port 3 output buffers can sink/source 4 LS TTL inputs. Port 3 pins that have 1s written to them are pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 3 pins that are externally being pulled low will source current (IIL, on the data sheet) because of the pullups.

Port 3 also serves the functions of various special features of the MCS-51 Family, as listed below:

| Port Pin | Alternative Function                   |
|----------|--|
| P3.0     | RXD (serial input port)                |
| P3.1     | TXD (serial output port)               |
| P3.2     | INT0 (external interrupt 0)            |
| P3.3     | INT1 (external interrupt 1)            |
| P3.4     | T0 (Timer 0 external input)            |
| P3.5     | T1 (Timer 1 external input)            |
| P3.6     | WR (external data memory write strobe) |
| P3.7     | RD (external data memory read strobe)  |



Reset input. A high on this pin for two machine cycles while the oscillator is running resets the device.

## ALE/PROG

Address Latch Enable output pulse for latching the low byte of the address during accesses to external memory. ALE can drive 8 LS TTL inputs. This pin is also the program pulse input (PROG) during programming of the EPROM parts.

In normal operation ALE is emitted at a constant rate of  $\frac{1}{6}$  the oscillator frequency, and may be used for external timing or clocking purposes. Note, however, that one ALE pulse is skipped during each access to external Data Memory.

## PSEN

Program Store Enable is the read strobe to external Program Memory. PSEN can drive 8 LS TTL inputs.

When the device is executing code from external Program Memory, PSEN is activated twice each machine cycle, except that two PSEN activations are skipped during each access to external Data Memory.

## EA/VPP

External Access enable EA must be externally held low in order to enable any MCS-51 device to fetch code from external Program Memory locations 0 to 0FFFF (0 to 1FFFFH, in the 8032AH and 8052AH).

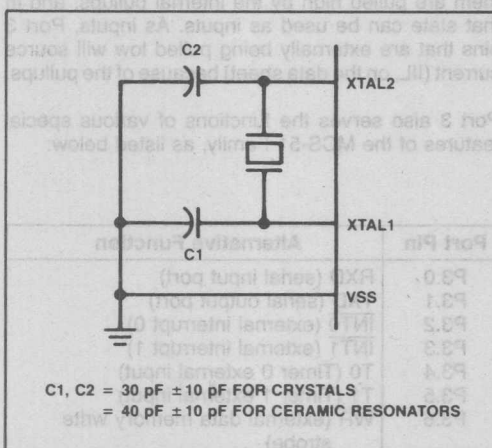


Figure 3. Oscillator Connections

Note, however, that if the Security Bit in the EPROM devices is programmed, the device will not fetch code from any location in external Program Memory.

This pin also receives the 21V programming supply voltage (VPP) during programming of the EPROM parts.

## XTAL1

Input to the inverting oscillator amplifier.

## XTAL2

Output from the inverting oscillator amplifier.

## OSCILLATOR CHARACTERISTICS

XTAL1 and XTAL2 are the input and output, respectively, of an inverting amplifier which can be configured for use as an on-chip oscillator, as shown in Figure 3. Either a quartz crystal or ceramic resonator may be used. More detailed information concerning the use of the on-chip oscillator is available in Application Note AP-155, "Oscillators for Microcontrollers."

To drive the device from an external clock source, XTAL1 should be grounded, while XTAL2 is driven, as shown in Figure 4. There are no requirements on the duty cycle of the external clock signal, since the input to the internal clocking circuitry is through a divide-by-two flip-flop, but minimum and maximum high and low times specified on the Data Sheet must be observed.

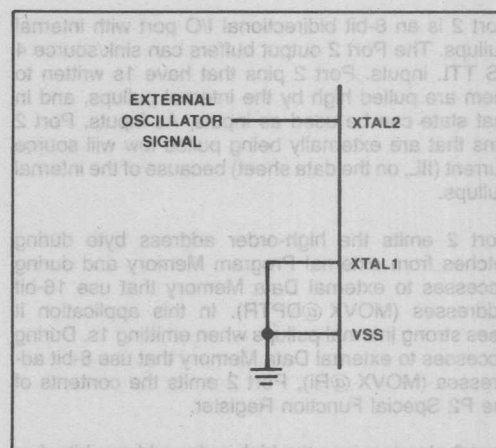


Figure 4. External Drive Configuration

# ABSOLUTE MAXIMUM RATINGS

Ambient Temperature Under Bias . . . 0 °C to 70 °C  
 Storage Temperature . . . -65 °C to +150 °C  
 Voltage on EA/VPP Pin to VSS . . -0.5V to +21.5V  
 Voltage on Any Other Pin to VSS . . -0.5V to +7V  
 Power Dissipation . . . . . 1.5W

**NOTICE:** Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

## D.C. CHARACTERISTICS: (TA = 0 °C to 70 °C; VCC = 5V ± 10%; VSS = 0V)

| Symbol | Parameter  | Min  | Max           | Unit     | Test Conditions                      |
|--------|--|------|---------------|----------|--------------------------------------|
| VIL    | Input Low Voltage (Except EA Pin of 8751H, 8751H-12 & 8751H-88)      | -0.5 | 0.8           | V        |                                      |
| VIL1   | Input Low Voltage to EA Pin of 8751H, 8751H-12 & 8751H-88            | 0    | 0.7           | V        |                                      |
| VIH    | Input High Voltage (Except XTAL2, RST)                               | 2.0  | VCC + 0.5     | V        |                                      |
| VIH1   | Input High Voltage to XTAL2, RST                                     | 2.5  | VCC + 0.5     | V        | XTAL1 = VSS                          |
| VOL    | Output Low Voltage (Ports 1, 2, 3)*                                  |      | 0.45          | V        | IOL = 1.6 mA                         |
| VOL1   | Output Low Voltage (Port 0, ALE, PSEN)*                              |      |               |          |                                      |
|        | 8751H, 8751H-12 & 8751H-88   |      | 0.60<br>0.45  | V<br>V   | IOL = 3.2 mA<br>IOL = 2.4 mA         |
|        | All Others   |      | 0.45          | V        | IOL = 3.2 mA                         |
| VOH    | Output High Voltage (Ports 1, 2, 3)                                  | 2.4  |               | V        | IOH = -80 µA                         |
| VOH1   | Output High Voltage (Port 0 in External Bus Mode, ALE, PSEN)         | 2.4  |               | V        | IOH = -400 µA                        |
| IIL    | Logical 0 Input Current (Ports 1, 2, 3 RST)                          |      |               |          |                                      |
|        | 8032AH, 8052AH<br>All Others   |      | -800<br>-500  | µA<br>µA | Vin = 0.45 V<br>Vin = 0.45 V         |
| IIL1   | Logical 0 Input Current to EA Pin of 8751H, 8751H-12 & 8751H-88 Only |      | -15           | mA       |                                      |
| IIL2   | Logical 0 Input Current (XTAL2)                                      |      | -3.2          | mA       | Vin = 0.45 V                         |
| ILI    | Input Leakage Current (Port 0)                                       |      |               |          |                                      |
|        | 8751H, 8751H-12 & 8751H-88<br>All Others                             |      | ± 100<br>± 10 | µA<br>µA | 0.45 < Vin < VCC<br>0.45 < Vin < VCC |
| IIH    | Logical 1 Input Current to EA Pin of 8751H, 8751H-12 & 8751H-88      |      | 500           | µA       |                                      |
| IIH1   | Input Current to RST to Activate Reset                               |      | 500           | µA       | Vin < (VCC - 1.5V)                   |
| ICC    | Power Supply Current: 8031/8051                                      |      | 160           | mA       | All Outputs Disconnected; EA = VCC   |
|        | 8031AH/8051AH  |      | 125           | mA       |                                      |
|        | 8032AH/8052AH  |      | 175           | mA       |                                      |
|        | 8751H/8751H-12/8751H-88  |      | 250           | mA       |                                      |
| CIO    | Pin Capacitance  |      | 10            | pF       | test freq = 1MHz                     |

\*Note: Capacitive loading on Ports 0 and 2 may cause spurious noise pulses to be superimposed on the VOLs of ALE and Ports 1 and 3. The noise is due to external bus capacitance discharging into the Port 0 and Port 2 pins when these pins make 1-to-0 transitions during bus operations. In the worst cases (capacitive loading > 100 pF), the noise pulse on the ALE line may exceed 0.8V. In such cases it may be desirable to qualify ALE with a Schmitt Trigger, or use an address latch with a Schmitt Trigger STROBE input.

**A.C. CHARACTERISTICS:** \* ( $T_A = 0^\circ\text{C}$  to  $+70^\circ\text{C}$ ,  $V_{CC} = 5V \pm 10\%$ ,  $V_{SS} = 0V$ ,  
Load Capacitance for Port 0, ALE, and PSEN = 100 pF,  
Load Capacitance for All Other Outputs = 80 pF)

| Symbol  | Parameter  | 12MHz Osc  |            | Variable Oscillator    |                          | Units    |
|---------|--|------------|------------|------------------------|--------------------------|----------|
|         |  | Min        | Max        | Min                    | Max                      |          |
| 1/TCLCL | Oscillator Frequency   |            |            | 3.5                    | 12.                      | MHz      |
| TLHLL   | ALE Pulse Width  | 127        |            | 2TCLCL-40              |                          | ns       |
| TAVLL   | Address Valid to ALE Low                                     | 43         |            | TCLCL-40               |                          | ns       |
| TLLAX   | Address Hold After ALE Low                                   | 48         |            | TCLCL-35               |                          | ns       |
| TLLIV   | ALE Low to Valid Instr In<br>8751H, 8751H-12<br>All Others   |            | 183<br>233 |                        | 4TCLCL-150<br>4TCLCL-100 | ns       |
| TLLPL   | ALE Low to PSEN Low  | 58         |            | TCLCL-25               |                          | ns       |
| TPLPH   | PSEN Pulse Width<br>8751H, 8751H-12<br>All Others            | 190<br>215 |            | 3TCLCL-60<br>3TCLCL-35 |                          | ns<br>ns |
| TPLIV   | PSEN Low to Valid Instr In<br>8751H, 8751H-12<br>All Others  |            | 100<br>125 |                        | 3TCLCL-150<br>3TCLCL-125 | ns<br>ns |
| TPXIX   | Input Instr Hold After PSEN                                  | 0          |            | 0                      |                          | ns       |
| TPXIZ   | Input Instr Float After PSEN                                 |            | 63         |                        | TCLCL-20                 | ns       |
| TPXAV   | PSEN to Address Valid  | 75         |            | TCLCL-8                |                          | ns       |
| TAVIV   | Address to Valid Instr In<br>8751H, 8751H-12<br>All Others   |            | 267<br>302 |                        | 5TCLCL-150<br>5TCLCL-115 | ns<br>ns |
| TPLAZ   | PSEN Low to Address Float                                    |            | 20         |                        | 20                       | ns       |
| TRLRH   | RD Pulse Width   | 400        |            | 6TCLCL-100             |                          | ns       |
| TWLWH   | WR Pulse Width   | 400        |            | 6TCLCL-100             |                          | ns       |
| TRLDV   | RD Low to Valid Data In                                      |            | 252        |                        | 5TCLCL-165               | ns       |
| TRHDX   | Data Hold After RD   | 0          |            | 0                      |                          | ns       |
| TRHDZ   | Data Float After RD  |            | 97         |                        | 2TCLCL-70                | ns       |
| TLLDV   | ALE Low to Valid Data In                                     |            | 517        |                        | 8TCLCL-150               | ns       |
| TAVDV   | Address to Valid Data In                                     |            | 585        |                        | 9TCLCL-165               | ns       |
| TLLWL   | ALE Low to RD or WR Low                                      | 200        | 300        | 3TCLCL-50              | 3TCLCL + 50              | ns       |
| TAVWL   | Address to RD or WR Low                                      | 203        |            | 4TCLCL-130             |                          | ns       |
| TQVWX   | Data Valid to WR Transition<br>8751H, 8751H-12<br>All Others | 13<br>23   |            | TCLCL-70<br>TCLCL-60   |                          | ns<br>ns |
| TQVWH   | Data Valid to WR High  | 433        |            | 7TCLCL-150             |                          | ns       |
| TWHQX   | Data Held After WR   | 33         |            | TCLCL-50               |                          | ns       |
| TRLAZ   | RD Low to Address Float                                      |            | 20         |                        | 20                       | ns       |
| TWHLH   | RD or WR High to ALE High<br>8751H, 8751H-12<br>All Others   | 33<br>43   | 133<br>123 | TCLCL-50<br>TCLCL-40   | TCLCL + 50<br>TCLCL + 40 | ns<br>ns |

\* This table does not include the 8751-88 AC characteristics (see next page).

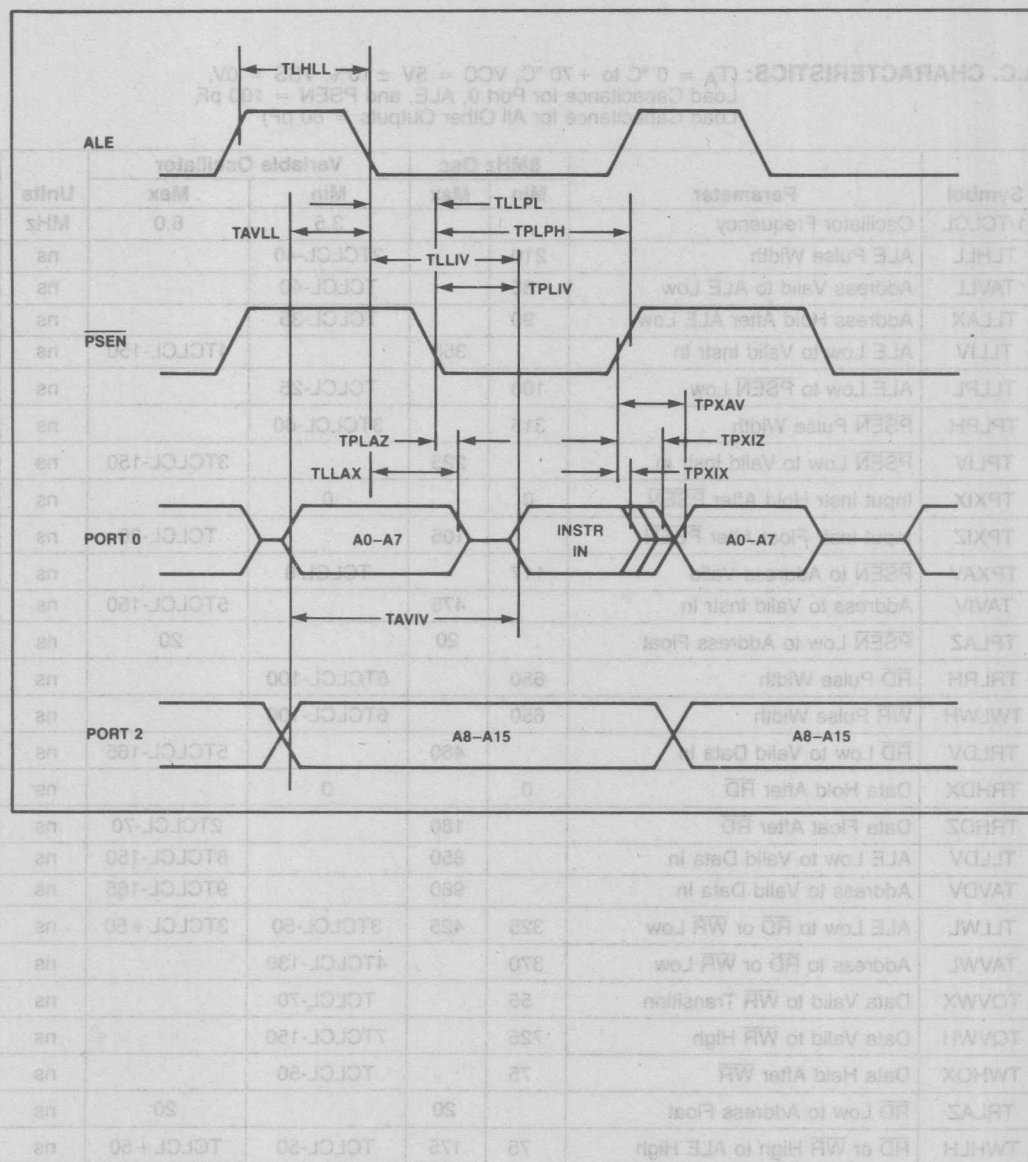
This Table is only for the 8751H-88

**A.C. CHARACTERISTICS:** ( $T_A = 0^\circ\text{C}$  to  $+70^\circ\text{C}$ ,  $V_{CC} = 5V \pm 10\%$ ,  $V_{SS} = 0V$ ,  
Load Capacitance for Port 0, ALE, and PSEN = 100 pF,  
Load Capacitance for All Other Outputs = 80 pF)

| Symbol  | Parameter   | 8MHz Osc |     | Variable Oscillator |             | Units |
|---------|---|----------|-----|---------------------|-------------|-------|
|         |   | Min      | Max | Min                 | Max         |       |
| 1/TCLCL | Oscillator Frequency                                |          |     | 3.5                 | 8.0         | MHz   |
| TLHLL   | ALE Pulse Width                                     | 210      |     | 2TCLCL-40           |             | ns    |
| TAVLL   | Address Valid to ALE Low                            | 85       |     | TCLCL-40            |             | ns    |
| TLLAX   | Address Hold After ALE Low                          | 90       |     | TCLCL-35            |             | ns    |
| TLLIV   | ALE Low to Valid Instr In                           |          | 350 |                     | 4TCLCL-150  | ns    |
| TLLPL   | ALE Low to PSEN Low                                 | 100      |     | TCLCL-25            |             | ns    |
| TPLPH   | PSEN Pulse Width                                    | 315      |     | 3TCLCL-60           |             | ns    |
| TPLIV   | PSEN Low to Valid Instr In                          |          | 225 |                     | 3TCLCL-150  | ns    |
| TPXIX   | Input Instr Hold After PSEN                         | 0        |     | 0                   |             | ns    |
| TPXIZ   | Input Instr Float After PSEN                        |          | 105 |                     | TCLCL-20    | ns    |
| TPXAV   | PSEN to Address Valid                               | 117      |     | TCLCL-8             |             | ns    |
| TAVIV   | Address to Valid Instr In                           |          | 475 |                     | 5TCLCL-150  | ns    |
| TPLAZ   | PSEN Low to Address Float                           |          | 20  |                     | 20          | ns    |
| TRLRH   | $\overline{RD}$ Pulse Width                         | 650      |     | 6TCLCL-100          |             | ns    |
| TWLWH   | $\overline{WR}$ Pulse Width                         | 650      |     | 6TCLCL-100          |             | ns    |
| TRLDV   | $\overline{RD}$ Low to Valid Data In                |          | 460 |                     | 5TCLCL-165  | ns    |
| TRHDX   | Data Hold After $\overline{RD}$                     | 0        |     | 0                   |             | ns    |
| TRHDZ   | Data Float After $\overline{RD}$                    |          | 180 |                     | 2TCLCL-70   | ns    |
| TLLDV   | ALE Low to Valid Data In                            |          | 850 |                     | 8TCLCL-150  | ns    |
| TAVDV   | Address to Valid Data In                            |          | 960 |                     | 9TCLCL-165  | ns    |
| TLLWL   | ALE Low to $\overline{RD}$ or $\overline{WR}$ Low   | 325      | 425 | 3TCLCL-50           | 3TCLCL + 50 | ns    |
| TAVWL   | Address to $\overline{RD}$ or $\overline{WR}$ Low   | 370      |     | 4TCLCL-130          |             | ns    |
| TQVWX   | Data Valid to $\overline{WR}$ Transition            | 55       |     | TCLCL-70            |             | ns    |
| TQVWH   | Data Valid to $\overline{WR}$ High                  | 725      |     | 7TCLCL-150          |             | ns    |
| TWHQX   | Data Held After $\overline{WR}$                     | 75       |     | TCLCL-50            |             | ns    |
| TRLAZ   | $\overline{RD}$ Low to Address Float                |          | 20  |                     | 20          | ns    |
| TWHLH   | $\overline{RD}$ or $\overline{WR}$ High to ALE High | 75       | 175 | TCLCL-50            | TCLCL + 50  | ns    |

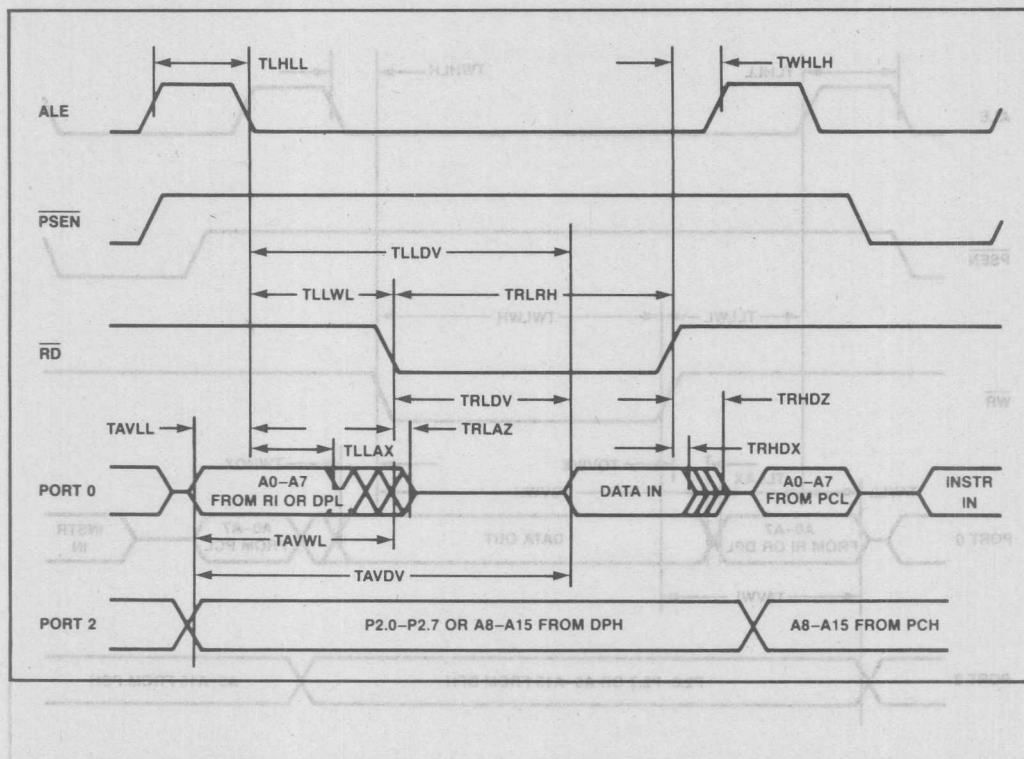


### EXTERNAL PROGRAM MEMORY READ CYCLE

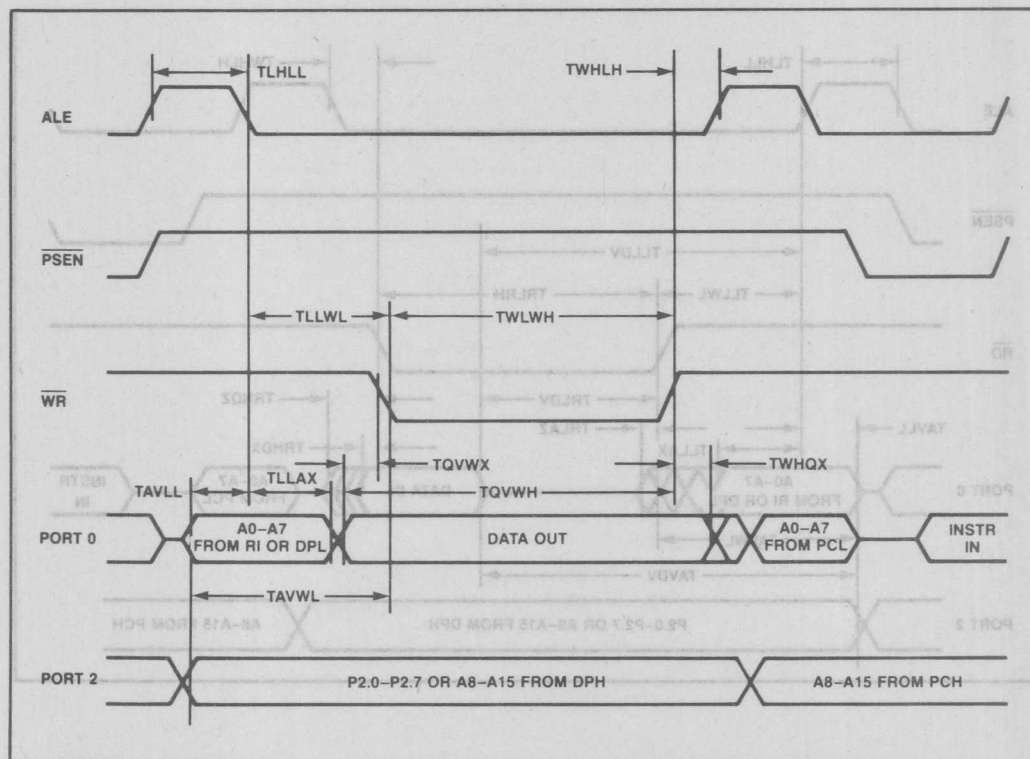




EXTERNAL DATA MEMORY READ CYCLE



EXTERNAL DATA MEMORY WRITE CYCLE

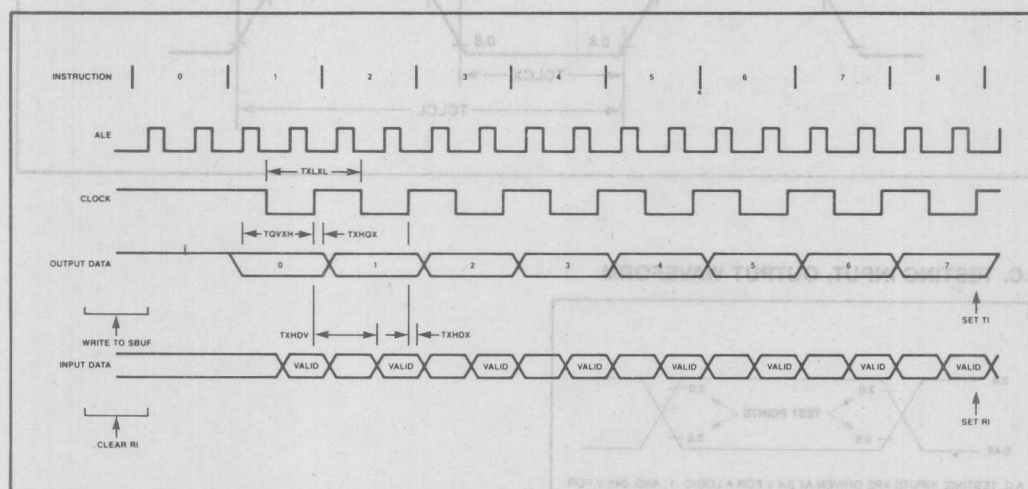


# SERIAL PORT TIMING — SHIFT REGISTER MODE

Test Conditions:  $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ;  $V_{CC} = 5V \pm 10\%$ ;  $V_{SS} = 0V$ ; Load Capacitance =  $80\text{ pF}$

| Symbol | Parameter                                | 12MHz Osc |     | Variable Oscillator |             | Units         |
|--------|--|-----------|-----|---------------------|-------------|---------------|
|        |  | Min       | Max | Min                 | Max         |               |
| TXLXL  | Serial Port Clock Cycle Time             | 1.0       |     | 12TCLCL             |             | $\mu\text{s}$ |
| TQVXH  | Output Data Setup to Clock Rising Edge   | 700       |     | 10TCLCL-133         |             | ns            |
| TXHQX  | Output Data Hold After Clock Rising Edge | 50        |     | 2TCLCL-117          |             | ns            |
| TXHDX  | Input Data Hold After Clock Rising Edge  | 0         |     | 0                   |             | ns            |
| TXHDV  | Clock Rising Edge to Input Data Valid    |           | 700 |                     | 10TCLCL-133 | ns            |

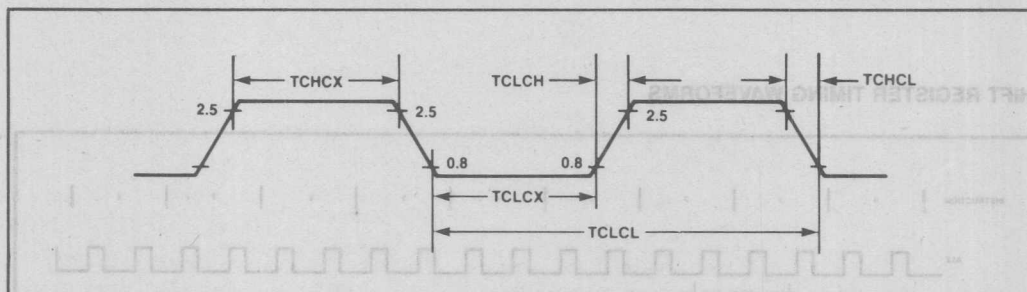
## SHIFT REGISTER TIMING WAVEFORMS



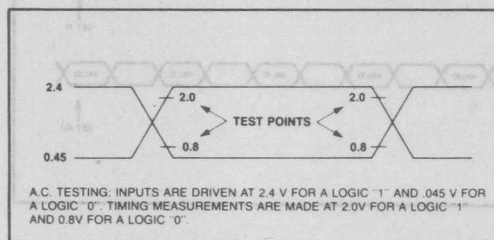
# EXTERNAL CLOCK DRIVE

| Symbol  | Parameter                              | Min | Max | Units |
|---------|--|-----|-----|-------|
| 1/TCLCL | Oscillator Frequency (except 8751H-88) | 3.5 | 12  | MHz   |
|         | 8751H-88                               | 3.5 | 8   | MHz   |
| TCHCX   | High Time                              | 20  |     | ns    |
| TCLCX   | Low Time                               | 20  |     | ns    |
| TCLCH   | Rise Time                              |     | 20  | ns    |
| TCHCL   | Fall Time                              |     | 20  | ns    |

## EXTERNAL CLOCK DRIVE WAVEFORMS



## A.C. TESTING INPUT, OUTPUT WAVEFORM



# EPROM CHARACTERISTICS:

Table 3. EPROM Programming Modes

| Mode         | RST | PSEN | ALE | EA  | P2.7 | P2.6 | P2.5 | P2.4 |
|--------------|-----|------|-----|-----|------|------|------|------|
| Program      | 1   | 0    | 0*  | VPP | 1    | 0    | X    | X    |
| Inhibit      | 1   | 0    | 1   | X   | 1    | 0    | X    | X    |
| Verify       | 1   | 0    | 1   | 1   | 0    | 0    | X    | X    |
| Security Set | 1   | 0    | 0*  | VPP | 1    | 1    | X    | X    |

Note: "1" = logic high for that pin  
"0" = logic low for that pin  
"X" = "don't care"  
"VPP" = +21V  $\pm$  0.5V

\*ALE is pulsed low for 50 msec.

## Programming the EPROM

To be programmed, the part must be running with a 4 to 6 MHz oscillator. (The reason the oscillator needs to be running is that the internal bus is being used to transfer address and program data to appropriate internal registers.) The address of an EPROM location to be programmed is applied to Port 1 and pins P2.0–P2.3 of Port 2, while the code byte to be programmed into that location is applied to Port 0. The other Port 2 pins, and RST, PSEN, and EA should be held at the "Program" levels indicated in Table 3. ALE is pulsed low for 50 msec to program the code byte into the addressed EPROM location. The setup is shown in Figure 5.

Normally EA is held at a logic high until just before ALE is to be pulsed. Then EA is raised to +21V, ALE is pulsed, and then EA is returned to a logic high. Waveforms and detailed timing specifications are shown in later sections of this data sheet.

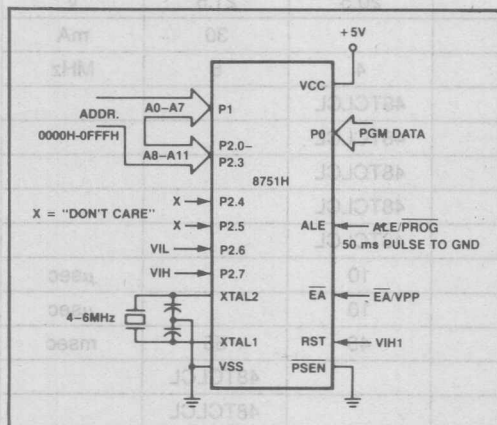


Figure 5. Programming Configuration

Note that the EA/VPP pin must not be allowed to go above the maximum specified VPP level of 21.5V for any amount of time. Even a narrow glitch above that voltage level can cause permanent damage to the device. The VPP source should be well regulated and free of glitches.

## Program Verification

If the Security Bit has not been programmed, the on-chip Program Memory can be read out for verification purposes, if desired, either during or after the programming operation. The address of the Program Memory location to be read is applied to Port 1 and pins P2.0–P2.3. The other pins should be held at the "Verify" levels indicated in Table 3. The contents of the addressed location will come out on Port 0. External pullups are required on Port 0 for this operation.

The setup, which is shown in Figure 6, is the same as for programming the EPROM except that pin P2.7 is held at a logic low, or may be used as an active-low read strobe.

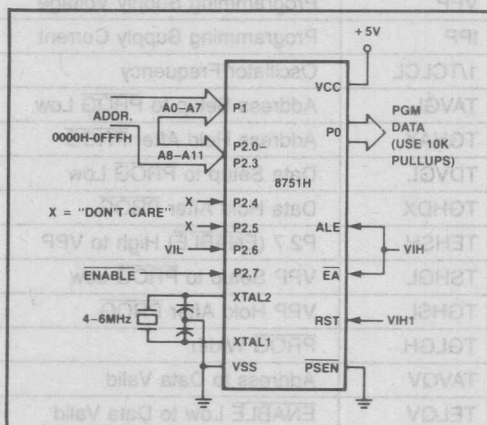


Figure 6. Program Verification



## EPROM Security

The security feature consists of a "locking" bit which when programmed denies electrical access by any external means to the on-chip Program Memory. The bit is programmed as shown in Figure 7. The setup and procedure are the same as for normal EPROM programming, except that P2.6 is held at a logic high. Port 0, Port 1, and pins P2.0–P2.3 may be in any state. The other pins should be held at the "Security" levels indicated in Table 3.

Once the Security Bit has been programmed, it can be cleared only by full erasure of the Program Memory. While it is programmed, the internal Program Memory can not be read out, the device can not be further programmed, and it **can not execute out of external program memory**. Erasing the EPROM, thus clearing the Security Bit, restores the device's full functionality. It can then be reprogrammed.

## Erasure Characteristics

Erasure of the EPROM begins to occur when the chip is exposed to light with wavelengths shorter than approximately 4,000 Angstroms. Since sunlight and fluorescent lighting have wavelengths in this range, exposure to these light sources over an extended time (about 1 week in sunlight, or 3 years in room-level fluorescent lighting) could cause inadvertent erasure. If an application subjects the device to this type of exposure, it is suggested that an opaque label be placed over the window.

## EPROM PROGRAMMING AND VERIFICATION CHARACTERISTICS:

(T<sub>A</sub> = 21 °C to 27 °C, VCC = 5V ± 10%, VSS = 0V)

| Symbol  | Parameter                                       | Min     | Max     | Units |
|---------|---|---------|---------|-------|
| VPP     | Programming Supply Voltage                      | 20.5    | 21.5    | V     |
| IPP     | Programming Supply Current                      |         | 30      | mA    |
| 1/TCLCL | Oscillator Frequency                            | 4       | 6       | MHz   |
| TAVGL   | Address Setup to $\overline{\text{PROG}}$ Low   | 48TCLCL |         |       |
| TGHAX   | Address Hold After $\overline{\text{PROG}}$     | 48TCLCL |         |       |
| TDVGL   | Data Setup to $\overline{\text{PROG}}$ Low      | 48TCLCL |         |       |
| TGHDX   | Data Hold After $\overline{\text{PROG}}$        | 48TCLCL |         |       |
| TEHSH   | P2.7 ( $\overline{\text{ENABLE}}$ ) High to VPP | 48TCLCL |         |       |
| TSHGL   | VPP Setup to $\overline{\text{PROG}}$ Low       | 10      |         | μsec  |
| TGHSL   | VPP Hold After $\overline{\text{PROG}}$         | 10      |         | μsec  |
| TGLGH   | $\overline{\text{PROG}}$ Width                  | 45      | 55      | msec  |
| TAVQV   | Address to Data Valid                           |         | 48TCLCL |       |
| TELQV   | $\overline{\text{ENABLE}}$ Low to Data Valid    |         | 48TCLCL |       |
| TEHQZ   | Data Float After $\overline{\text{ENABLE}}$     | 0       | 48TCLCL |       |

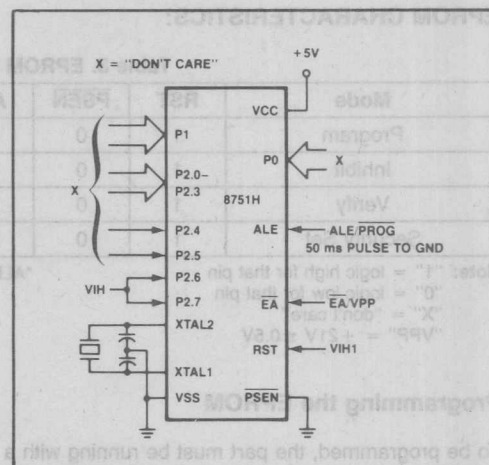
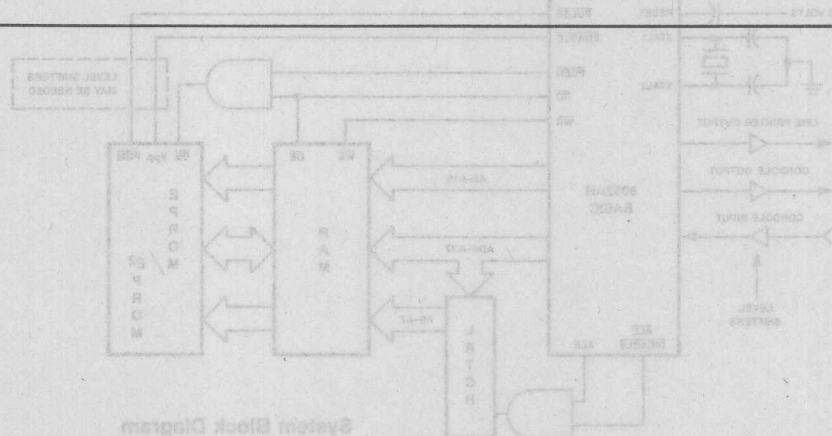
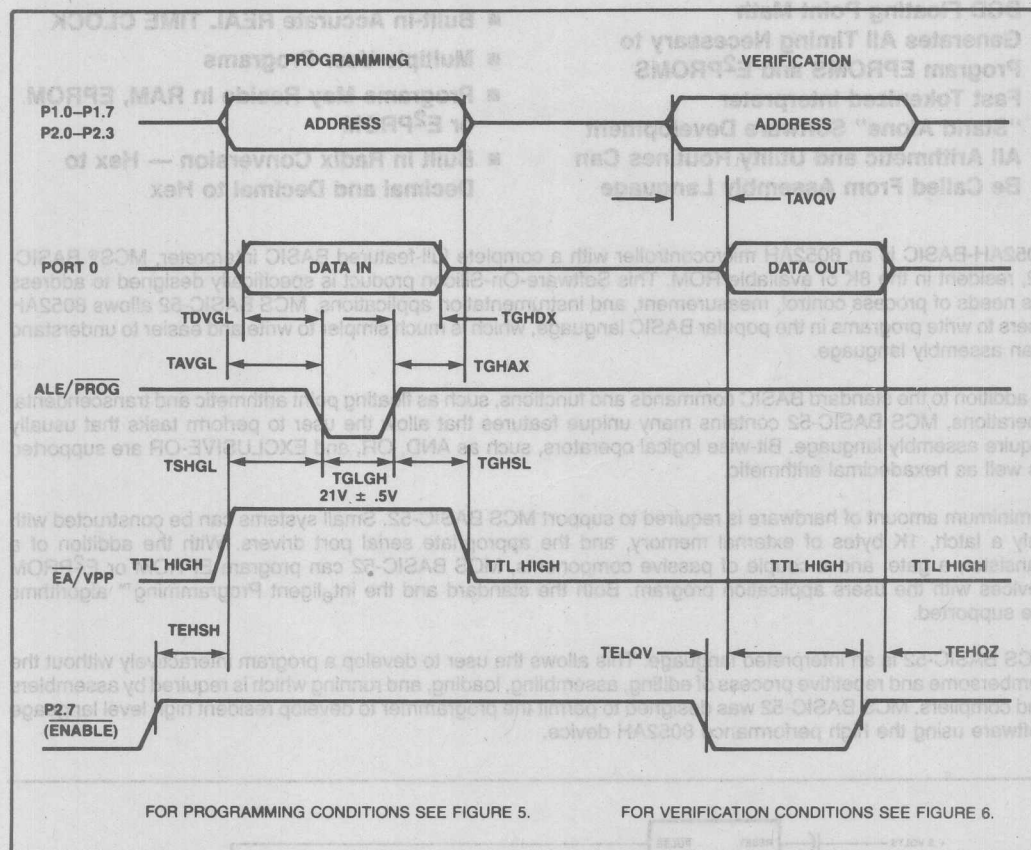


Figure 7. Programming the Security Bit

The recommended erasure procedure is exposure to ultraviolet light (at 2537 Angstroms) to an integrated dose of at least 15 W-sec/cm<sup>2</sup>. Exposing the EPROM to an ultraviolet lamp of 12,000 μW/cm<sup>2</sup> rating for 20 to 30 minutes, at a distance of about 1 inch, should be sufficient.

Erasure leaves the array in an all 1s state.

# EPROM PROGRAMMING AND VERIFICATION WAVEFORMS



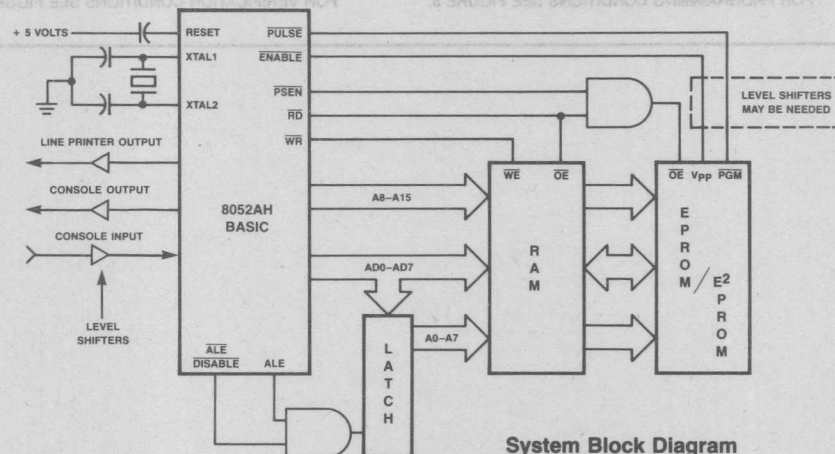
- BCD Floating Point Math
- Generates All Timing Necessary to Program EPROMS and E<sup>2</sup>PROMS
- Fast Tokenized Interpreter
- "Stand Alone" Software Development
- All Arithmetic and Utility Routines Can Be Called From Assembly Language
- Built-In Accurate REAL TIME CLOCK
- Multiple User Programs
- Programs May Reside in RAM, EPROM or E<sup>2</sup>PROM
- Built in Radix Conversion — Hex to Decimal and Decimal to Hex

8052AH-BASIC is an 8052AH microcontroller with a complete full-featured BASIC interpreter, MCS® BASIC-52, resident in the 8K of available ROM. This Software-On-Silicon product is specifically designed to address the needs of process control, measurement, and instrumentation applications. MCS BASIC-52 allows 8052AH users to write programs in the popular BASIC language, which is much simpler to write and easier to understand than assembly language.

In addition to the standard BASIC commands and functions, such as floating point arithmetic and transcendental operations, MCS BASIC-52 contains many unique features that allow the user to perform tasks that usually require assembly language. Bit-wise logical operators, such as AND, OR, and EXCLUSIVE-OR are supported as well as hexadecimal arithmetic.

A minimum amount of hardware is required to support MCS BASIC-52. Small systems can be constructed with only a latch, 1K bytes of external memory, and the appropriate serial port drivers. With the addition of a transistor, a gate, and a couple of passive components, MCS BASIC-52 can program EPROM or E<sup>2</sup>PROM devices with the users application program. Both the standard and the intelligent Programming™ algorithms are supported.

MCS BASIC-52 is an interpreted language. This allows the user to develop a program interactively without the cumbersome and repetitive process of editing, assembling, loading, and running which is required by assemblers and compilers. MCS BASIC-52 was designed to permit the programmer to develop resident high level language software using the high performance 8052AH device.



## COMMAND SET

MCS BASIC-52 contains all standard BASIC commands, statements, and operators. Figure 1 list the software feature set of MCS BASIC-52.

## DATA FORMAT

The range of numbers that can be represented in MCS BASIC-52 is:

$$\pm 1E-127 \text{ to } \pm .99999999E + 127$$

## CONTROL ORIENTED FEATURES

MCS BASIC-52 contains many unique features to perform task that usually require assembly language programming. The XBY and DBY operators can read and/or write external and internal memory respectively. The CBY operator is used to read program memory. Additionally, virtually all of the special function registers on the 8052AH can be accessed with MCS BASIC-52. This allows the user to set the timer or interrupt modes within the constructs of a BASIC program. An accurate interrupt driven REAL TIME CLOCK that has a 5 millisecond resolution is also implemented in MCS BASIC-52. This clock can be enabled, disabled, and used to generate interrupts. Finally, a CALL statement that allows the programmer to CALL assembly language routines is available in MCS BASIC-52. Parameters can be passed in a number of different ways.

## EPROM/E<sup>2</sup>PROM FILE

Most Basic interpreters allow only one program to be resident in memory, and many require that the program reside in RAM. MCS BASIC-52 allows programs to reside in both RAM and EPROM/E<sup>2</sup>PROM. Additionally, up to 255 programs may reside in EPROM/E<sup>2</sup>PROM. Programs may also be transferred (XFER) from EPROM/E<sup>2</sup>PROM to RAM for editing purposes.

## EPROM/E<sup>2</sup>PROM PROGRAMMING

A powerful feature of MCS BASIC-52 is that it generates all of the timing necessary to program any standard EPROM or E<sup>2</sup>PROM device with the users' program (PROG/FPROG). Additionally, very little external hardware is required to implement this feature. Saving programs in EPROM/E<sup>2</sup>PROM is much more attractive and reliable than other alternatives, such as cassette tape, especially in control and/or other noisy environments.

After the user programs an EPROM or E<sup>2</sup>PROM with the desired BASIC program. The PROG2 or FPROG2 commands may be used to enable the unique AUTOSTART feature of MCS BASIC-52. When AUTOSTART is enabled, MCS BASIC-52 will execute the user program after RESET or a power-up condition. This permits the user to RUN a program without connecting the MCS BASIC-52 device to a console — a powerful feature for control environments.

## USER ACCESSABLE FUNCTION LIBRARY

Another unique feature of MCS BASIC-52 is that it contains a complete library of functions that can be accessed with assembly language. All floating point, radix conversion, and I/O routines contained in MCS BASIC-52 can be accessed with assembly language CALL instructions. These complex arithmetic routines can be used by the programmer in applications requiring the speed of assembly language, but also the complex arithmetics offered by BASIC.

## 8052AH-BASIC PIN DESCRIPTION (FIGURE 2)

8052AH-BASIC is an 8052AH device, however, MCS BASIC-52 assumes a particular hardware configuration. The following pin description outlines the pin functions defined by MCS BASIC-52.

### VSS

Circuit ground potential.

### VCC

Circuit supply voltage. 5 volts  $\pm$  10% relative to VSS.

### AD0-AD7

The multiplexed low-order address and data bus used during accesses to external memory. External pullup devices ( $\sim 10K \Omega$ ) are required on these pins if the MCS BASIC-52 EPROM/E<sup>2</sup>PROM programming feature is used.

### A8-A15

The high order address bus used during accesses to external memory.



| Commands | Statements   | Operators            |
|----------|--------------|----------------------|
| RUN      | BAUD         | ADD (+)              |
| LIST     | CALL         | DIVIDE (/)           |
| LIST#    | CLEAR        | EXPONENTIATION (**)  |
| NEW      | CLEAR        | MULTIPLY (*)         |
| NULL     | CLEAR        | SUBTRACT (-)         |
| RAM      | CLOCK0       | LOGICAL AND (.AND.)  |
| ROM      | CLOCK1       | LOGICAL OR (.OR.)    |
| XFER     | DATA         | LOGICAL X-OR (.XOR.) |
| PROG     | READ         | LOGICAL NOT          |
| PROG1    | RESTORE      | ABS ( )              |
| PROG2    | DIM          | INT ( )              |
| FPROG    | DO-WHILE     | SGN ( )              |
| FPROG1   | DO-UNTIL     | SQR ( )              |
| FPROG2   | END          | RND                  |
|          | FOR-TO-STEP  | LOG ( )              |
|          | NEXT         | EXP ( )              |
|          | GOSUB        | SIN ( )              |
|          | RETURN       | COS ( )              |
|          | GOTO         | TAN ( )              |
|          | ON-GOTO      | ATN ( )              |
|          | ON-GOSUB     | =, >, >=, <, <=, <>  |
|          | IF-THEN-ELSE | ASC ( )              |
|          | INPUT        | CHR ( )              |
|          | LET          | CBY ( )              |
|          | ONERR        | DBY ( )              |
|          | ONEXT1       | XBY ( )              |
|          | ONTIME       | GET                  |
|          | PRINT        | IE                   |
|          | PRINT#       | IP                   |
|          | PH0.#        | PORT1                |
|          | PH0.#        | PCON                 |
|          | PH1.         | RCAP2                |
|          | PH1.#        | T2CON                |
|          | PUSH         | TCON                 |
|          | POP          | TMOD                 |
|          | PWM          | TIME                 |
|          | REM          | TIMER0               |
|          | RET1         | TIMER1               |
|          | STOP         | TIMER2               |
|          | STRING       | TIME                 |
|          | UI0          | XTAL                 |
|          | UI1          | MTOP                 |
|          | UO0          | LEN                  |
|          | UO1          | FREE                 |
|          |              | PI                   |

Figure 1. MCS® BASIC-52 Software Feature Set

## PORT 1

A general purpose quasi-bidirectional 8-bit input/output port. The individual pins on PORT 1 all have alternate functions which may or may not be implemented by the user. The alternate functions are as follows:

### PORT 1.0 (T2)

Can be used as the trigger input to TIMER/COUNTER 2. A one (1) must be written to this port pin output latch in order for this function to operate. Details of

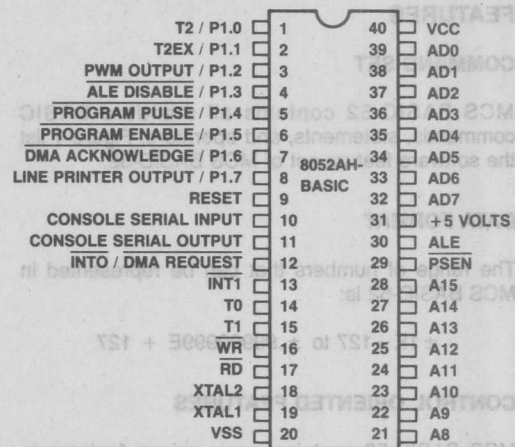


Figure 2. Configuration

the T2 trigger function are covered in the Microcontrollers Handbook, Order Number 210918-002.

### PORT 1.1 (T2EX)

Can be used as the external input to TIMER/COUNTER 2. A one (1) must be written to this port pin output latch in order for this function to operate. Details of the T2 trigger function are covered in the Microcontroller Users Manual.

### PORT 1.2 (PWM OUTPUT)

This pin is used as the PWM output port when the PWM statement is executed. PWM stands for Pulse Width Modulation and is used to generate pulses of varying duty cycle and frequency.

### PORT 1.3 (ALE DISABLE)

This pin is used to disable the ALE signal to the external address latch when the EPROM/E<sup>2</sup>PROM programming feature is used. In a system, this pin is logically anded with ALE.

### PORT 1.4 (PROGRAMMING PULSE)

When the EPROM/E<sup>2</sup>PROM programming feature is used, this pin provides the proper programming pulse width to program EPROM and INTELlIGENT EPROM™ devices. MCS BASIC-52 actually calculates the proper programming pulse width from the system crystal value (XTAL) to assure the proper timing of this pulse. When used to program E<sup>2</sup>PROM devices, the length of this pulse is not critical. This pin is active in the logical zero (0) state.



**PORT 1.5 (PROGRAMMING ENABLE)**

When the EPROM/E<sup>2</sup>PROM programming feature is implemented, this pin is used to enable the EPROM programming voltage. This pin remains active (logically low (0)) during the entire EPROM programming process. On E<sup>2</sup>PROM devices that do not require any special programming voltage, this pin is not used.

**PORT 1.6 (DMA ACKNOWLEDGE)**

When the DMA feature is implemented as described in the MCS® BASIC-52 users manual, this pin functions as an active low DMA ACKNOWLEDGE output.

**PORT 1.7 (LINE PRINTER OUTPUT)**

This pin functions as a serial output port when the LIST# or PRINT# command and/or statement is used. This enables the user to make a "hard copy" of a program or to print out results of a calculation.

**RESET**

A high (2.5 volts) on this pin for two machine cycles while the oscillator is running resets the device. An external pulldown resistor (~8.2K) from RESET to VSS permits power-on reset when a capacitor (~10 uf) is connected from this pin to VCC.

**ALE**

ALE (address latch enable) is an output pin that is used to latch the low order address byte during Read, Write, or program fetch operations to external memory.

**PSEN**

This pin (Program Store ENable) is a control signal that is used to enable external program memory. In MCS® BASIC-52, this pin will always remain inactive (logically high (1)) unless the user is running an assembly language program in external memory.

**XTAL1**

Input to the inverting amplifier that forms the oscillator.

**XTAL2**

Output of the inverting amplifier that forms the oscillator, and input to the internal clock generator. Receives the external oscillator signal when an external oscillator is used.

**RD**

A control signal that is used to enable READ operations to external data memory. This pin is active low (0).

**WR**

A control signal that is used to enable WRITE operations to external data memory. This pin is active low (0).

**T1**

This pin can be programmed to be an external input to TIMER/COUNTER 1.

**T0**

This pin can be programmed to be an external input to TIMER/COUNTER 0.

**INT1**

This pin is the external interrupt 1 pin. It is active low and interrupts on this pin may be handled in either BASIC or in assembly language.

**INT0/DMA REQUEST**

This is the external interrupt 0 pin. It is active low and may be optionally programmed to function as a DMA request input pin. The DMA REQUEST pin is used by E<sup>2</sup>PROM devices during programming.

**CONSOLE SERIAL OUTPUT**

This is the serial output pin to the console device. Standard ASCII codes are used as well as a standard asynchronous frame.

**CONSOLE SERIAL INPUT**

This is the serial input pin that receives data from the console device. Standard ASCII codes are assumed to be the input and the data is assumed to be transmitted using a standard asynchronous frame.

**NOTES**

If pin 31 is grounded the 8052AH-BASIC will operate as a standard 8032AH. The tolerances on this pin are described under DC characteristics.

For detailed information concerning this product please refer to the MCS BASIC-52 Users Manual (Order Number 210918-002).



Ambient Temperature Under Bias . . . . . 0°C to 70°C

Storage Temperature . . . . . -65°C to +150°C

Voltage on Any Pin With

Respect to Ground ( $V_{SS}$ ) . . . . . -0.5V to +7V

Power Dissipation . . . . . 2 Watts

Absolute maximum ratings may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

#### DC CHARACTERISTICS ( $T_A = 0^\circ\text{C}$ to $70^\circ\text{C}$ , $V_{CC} = 4.5\text{V}$ to $5.5\text{V}$ , $V_{SS} = 0\text{V}$ )

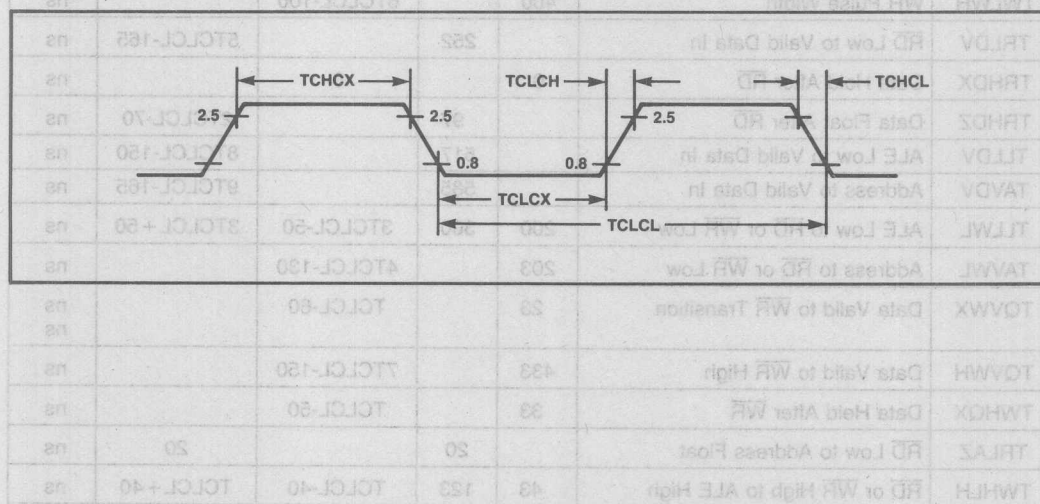
| Symbol | Parameter  | Min  | Max            | Unit    | Test Conditions                                |
|--------|--|------|----------------|---------|--|
| VIL    | Input Low Voltage  | -0.5 | 0.8            | V       |  |
| VIH    | Input High Voltage<br>(Except RST and XTAL2)               | 2.0  | $V_{CC} + 0.5$ | V       |  |
| VIH1   | Input High Voltage to<br>RST for Reset, XTAL2              | 2.5  | $V_{CC} + 0.5$ | V       | XTAL1 to $V_{SS}$                              |
| VOL    | Output Low Voltage Port 1, A8-15,<br>Control Functions     |      | 0.45           | V       | IOL = 1.6mA                                    |
| VOL1   | Output Low Voltage ALE, PSEN<br>(Note 1)                   |      | 0.45           | V       | IOL = 3.2mA                                    |
| VOH    | Output High Voltage Port 1, A8-15,<br>Control Functions    | 2.4  |                | V       | IOH = -80 $\mu$ A                              |
| VOH1   | Output High Voltage AD0-7, ALE, $\overline{\text{PSEN}}$   | 2.4  |                | V       | IOH = -400 $\mu$ A                             |
| IIL    | Logical 0 Input Current Port 1, A8-15<br>Control Functions |      | -800           | $\mu$ A | Vin = 0.45V                                    |
| IIL2   | Logical 0 Input Current XTAL2                              |      | -2.5           | mA      | XTAL1 at $V_{SS}$ ,<br>Vin = 0.45V             |
| IL1    | Input Leakage Current To AD0-7 EA                          |      | $\pm 10$       | $\mu$ A | $0.45\text{V} < V_{in} < V_{CC}$               |
| IIH1   | Input High Current to RST/VPD For<br>Reset                 |      | 500            | $\mu$ A | Vin = $V_{CC} - 1.5\text{V}$                   |
| ICC    | Power Supply Current                                       |      | 175            | mA      | All outputs disconnected                       |
| CIO    | Capacitance of I/O Buffer                                  |      | 10             | pF      | $f_c = 1\text{MHz}$ , $T_A = 25^\circ\text{C}$ |

**Note 1:** Vol is degraded when the 8032AH/8052AH rapidly discharges external capacitance. This AC noise is most pronounced during emission of address data. When using external memory, locate the latch or buffer as close to the 8032AH/8052AH as possible.

| Datum      | Emitting Ports | Degraded I/O Lines         | VOL (peak) (max) |
|------------|----------------|----------------------------|------------------|
| Address    | A8-15, AD0-7   | P1, Control Functions      | 0.8V             |
| Write Data | AD0-7          | P1, Control Functions, ALE | 0.8v             |

## EXTERNAL CLOCK DRIVE CHARACTERISTICS (XTAL2)

| Symbol | Parameter         | Variable Clock<br>f = 3.5 MHz to 12 MHz |     | Unit |
|--------|-------------------|---|-----|------|
|        |                   | Min                                     | Max |      |
| TCLCL  | Oscillator Period | 83.3                                    | 286 | ns   |
| TCHCX  | High Time         | 20                                      |     | ns   |
| TCLCX  | Low Time          | 20                                      |     | ns   |
| TCLCH  | Rise Time         |   | 20  | ns   |
| TCHCL  | Fall Time         |   | 20  | ns   |

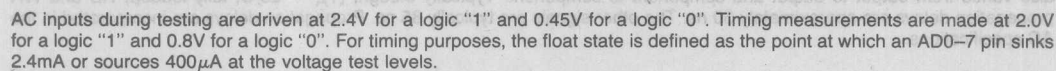


**A.C. CHARACTERISTICS:** ( $T_A = 0^\circ\text{C}$  to  $+70^\circ\text{C}$ ,  $V_{CC} = 5V \pm 10\%$ ,  $V_{SS} = 0V$ ,  
Load Capacitance for Port 0, ALE, and PSEN = 100 pF,  
Load Capacitance for All Other Outputs = 80 pF)

| Symbol  | Parameter   | 12MHz Osc |     | Variable Oscillator |             | Units    |
|---------|---|-----------|-----|---------------------|-------------|----------|
|         |   | Min       | Max | Min                 | Max         |          |
| 1/TCLCL | Oscillator Frequency  |           |     | 3.5                 | 12.         | MHz      |
| TLHLL   | ALE Pulse Width   | 127       |     | 2TCLCL-40           |             | ns       |
| TAVLL   | Address Valid to ALE Low  | 43        |     | TCLCL-40            |             | ns       |
| TLLAX   | Address Hold After ALE Low  | 48        |     | TCLCL-35            |             | ns       |
| TLLIV   | ALE Low to Valid Instr In   |           | 233 |                     | 4TCLCL-100  | ns       |
| TLLPL   | ALE Low to $\overline{\text{PSEN}}$ Low                           | 58        |     | TCLCL-25            |             | ns       |
| TPLPH   | $\overline{\text{PSEN}}$ Pulse Width                              | 215       |     | 3TCLCL-35           |             | ns       |
| TPLIV   | $\overline{\text{PSEN}}$ Low to Valid Instr In                    |           | 125 |                     | 3TCLCL-125  | ns       |
| TPXIX   | Input Instr Hold After $\overline{\text{PSEN}}$                   | 0         |     | 0                   |             | ns       |
| TPXIZ   | Input Instr Float After $\overline{\text{PSEN}}$                  |           | 63  |                     | TCLCL-20    | ns       |
| TPXAV   | $\overline{\text{PSEN}}$ to Address Valid                         | 75        |     | TCLCL-8             |             | ns       |
| TAVIV   | Address to Valid Instr In   |           | 302 |                     | 5TCLCL-115  | ns       |
| TPLAZ   | $\overline{\text{PSEN}}$ Low to Address Float                     |           | 20  |                     | 20          | ns       |
| TRLRH   | $\overline{\text{RD}}$ Pulse Width                                | 400       |     | 6TCLCL-100          |             | ns       |
| TWLWH   | $\overline{\text{WR}}$ Pulse Width                                | 400       |     | 6TCLCL-100          |             | ns       |
| TRLDV   | $\overline{\text{RD}}$ Low to Valid Data In                       |           | 252 |                     | 5TCLCL-165  | ns       |
| TRHDX   | Data Hold After $\overline{\text{RD}}$                            | 0         |     | 0                   |             | ns       |
| TRHDZ   | Data Float After $\overline{\text{RD}}$                           |           | 97  |                     | 2TCLCL-70   | ns       |
| TLLDV   | ALE Low to Valid Data In  |           | 517 |                     | 8TCLCL-150  | ns       |
| TAVDV   | Address to Valid Data In  |           | 585 |                     | 9TCLCL-165  | ns       |
| TLLWL   | ALE Low to $\overline{\text{RD}}$ or $\overline{\text{WR}}$ Low   | 200       | 300 | 3TCLCL-50           | 3TCLCL + 50 | ns       |
| TAVWL   | Address to $\overline{\text{RD}}$ or $\overline{\text{WR}}$ Low   | 203       |     | 4TCLCL-130          |             | ns       |
| TQVWX   | Data Valid to $\overline{\text{WR}}$ Transition                   | 23        |     | TCLCL-60            |             | ns<br>ns |
| TQVWH   | Data Valid to $\overline{\text{WR}}$ High                         | 433       |     | 7TCLCL-150          |             | ns       |
| TWHQX   | Data Held After $\overline{\text{WR}}$                            | 33        |     | TCLCL-50            |             | ns       |
| TRLAZ   | $\overline{\text{RD}}$ Low to Address Float                       |           | 20  |                     | 20          | ns       |
| TWHLH   | $\overline{\text{RD}}$ or $\overline{\text{WR}}$ High to ALE High | 43        | 123 | TCLCL-40            | TCLCL + 40  | ns       |

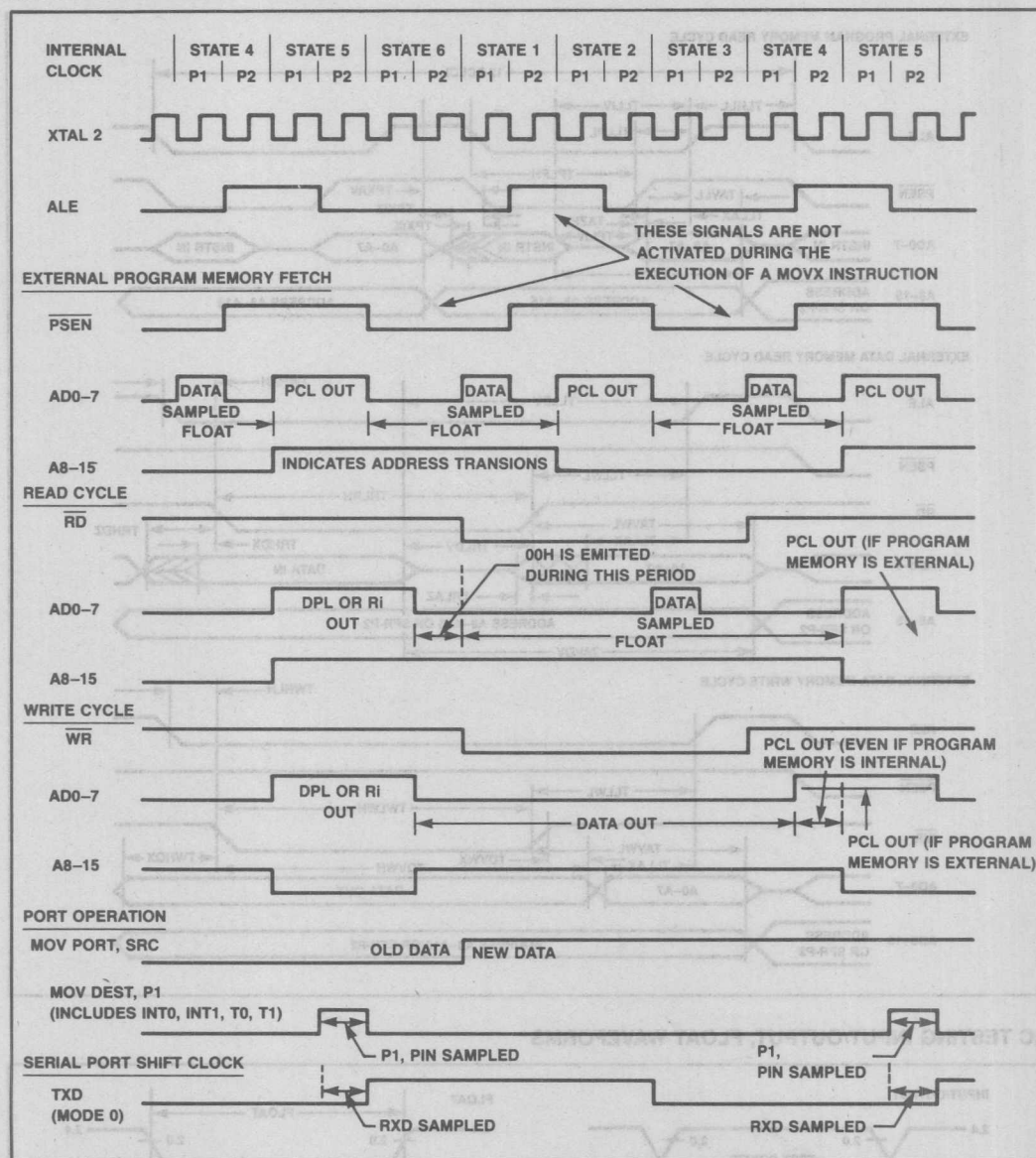


## NOV 13 1964 12:07 PM





# CLOCK WAVEFORMS



This diagram indicates when signals are clocked internally. The time it takes the signals to propagate to the pins, however, ranges from 25 to 125 ns. This propagation delay is dependent on variables such as temperature and pin loading. Propagation also varies from output to output and component to component. Typically though, ( $T_A = 25^\circ\text{C}$ , fully loaded) RD and WR propagation delays are approximately 50 ns. The other signals are typically 85 ns. Propagation delays are incorporated in the AC specifications.

# 80C31BH/80C31BH-1/80C31BH-2

## CHMOS SINGLE-CHIP 8-BIT CONTROL-ORIENTED CPU WITH RAM AND I/O

80C51BH/80C31BH — 3.5 TO 12 MHz,  $V_{CC} = 5\text{ V} \pm 20\%$ 

80C51BH-1/80C31BH-1 — 3.5 TO 16 MHz,  $V_{CC} = 5\text{ V} \pm 20\%$

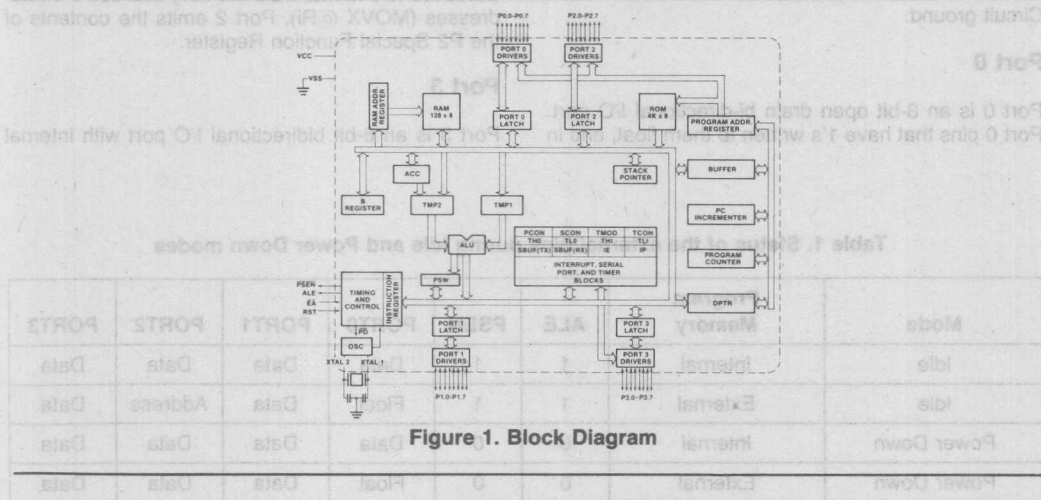
80C51BH-2/80C31BH-2 — 0.5 to 12 MHz,  $V_{CC} = 5\text{ V} \pm 20\%$ 

- Power Control Modes
- 128 x 8-Bit RAM
- 32 Programmable I/O Lines
- Two 16-Bit Timer/Counters
- 64K Program Memory Space
- High Performance CMOS Process
- Boolean Processor
- 5 Interrupt Sources
- Programmable Serial Port
- 64K Data Memory Space

The MCS®-51 CHMOS products are fabricated on Intel's CHMOS III process and are functionally compatible with the standard MCS-51 HMOS and EPROM products. CHMOS III is a technology which combines the high speed and density characteristics of HMOS with the low power attributes of CHMOS. This combination expands the effectiveness of the powerful MCS-51 architecture and instruction set.

Like the MCS-51 HMOS versions, the MCS-51 CHMOS products have the following features: 4K byte of ROM (80C51BH/80C51BH-1/80C51BH-2 only); 128 bytes of RAM; 32 I/O lines; two 16-bit timer/counters; a five-source two-level interrupt structure; a full duplex serial port; and on-chip oscillator and clock circuitry. In addition, the MCS-51 CHMOS products have two software selectable modes of reduced activity for further power reduction — Idle and Power Down.

The Idle mode freezes the CPU while allowing the RAM, timer/counters serial port and interrupt system to continue functioning. The Power Down mode saves the RAM contents but freezes the oscillator, causing all other chip functions to be inoperative.



## IDLE MODE

In the idle mode, the CPU puts itself to sleep while all the on chip peripherals stay active. The instruction that invokes the Idle mode is the last instruction executed in the normal operating mode before Idle mode is activated. The content of CPU, the on chip RAM, and all the Special Function Registers remain intact during this mode. The Idle mode can be terminated either by any enabled interrupt, at which time the process is picked up at the interrupt service routine and continued, or by a hardware reset which starts the processor the same as a power on reset.

## POWER DOWN MODE

In the Power Down mode the oscillator is stopped, and the instruction that invokes Power Down is the last instruction executed. Only the contents of the on chip RAM is preserved. A hardware reset is the only way to terminate Power Down.

The control bits for the reduced power modes are in the Special Function Register PCON.

**NOTE:** For more detailed information on these reduced power modes refer to the 1985 Microcontroller Handbook.

## PIN DESCRIPTIONS

### VCC

Supply voltage during normal, Idle, and Power Down operations.

### VSS

Circuit ground.

### Port 0

Port 0 is an 8-bit open drain bi-directional I/O port. Port 0 pins that have 1's written to them float, and in

that state can be used as high-impedance inputs.

Port 0 is also the multiplexed low-order address and data bus during accesses to external Program and Data Memory. In this application it uses strong internal pullups when emitting 1s. Port 0 also outputs the code bytes during program verification in the 80C51BH. External pullups are required during program verification.

### Port 1

Port 1 is an 8-bit bidirectional I/O port with internal pullups. Port 1 pins that have 1s written to them are pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 1 pins that are externally being pulled low will source current (IIL, on the data sheet) because of the internal pullups.

Port 1 also receives the low-order address bytes during program verification.

### Port 2

Port 2 is an 8-bit bidirectional I/O port with internal pullups. Port 2 pins that have 1s written to them are pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 2 pins that are externally being pulled low will source current (IIL, on the data sheet) because of the internal pullups.

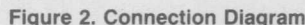
Port 2 emits the high-order address byte during fetches from external Program Memory and during accesses to external Data Memory that use 16-bit addresses (MOVX @DPTR). In this application it uses strong internal pullups when emitting 1s. During accesses to external Data Memory that use 8-bit addresses (MOVX @Ri), Port 2 emits the contents of the P2 Special Function Register.

### Port 3

Port 3 is an 8-bit bidirectional I/O port with internal

Table 1. Status of the external pins during Idle and Power Down modes

| Mode       | Program Memory | ALE | PSEN | PORT0 | PORT1 | PORT2   | PORT3 |
|------------|----------------|-----|------|-------|-------|---------|-------|
| Idle       | Internal       | 1   | 1    | Data  | Data  | Data    | Data  |
| Idle       | External       | 1   | 1    | Float | Data  | Address | Data  |
| Power Down | Internal       | 0   | 0    | Data  | Data  | Data    | Data  |
| Power Down | External       | 0   | 0    | Float | Data  | Data    | Data  |



Program Store Enable is the read strobe to external Program Memory.



When the 80C51BH is executing code from external Program Memory,  $\overline{\text{PSEN}}$  is activated twice each machine cycle, except that two  $\overline{\text{PSEN}}$  activations are skipped during each access to external Data Memory.  $\overline{\text{PSEN}}$  is not activated during fetches from internal program memory.

## EA

External Access enable.  $\overline{\text{EA}}$  must be externally held low in order to enable the device to fetch code from external Program Memory locations 0000H to 0FFFH. If  $\overline{\text{EA}}$  is held high the device executes from internal Program Memory unless the program counter contains an address greater than 0FFFH.

## XTAL1

Input to the inverting oscillator amplifier and input to the internal block generator circuits.

## XTAL2

Output from the inverting oscillator amplifier.

## OSCILLATOR CHARACTERISTICS

XTAL1 and XTAL2 are the input and output, respectively, of an inverting amplifier which can be configured for use as an on-chip oscillator, as shown in Figure 3. More detailed information concerning the use of the on-chip oscillator is available in Application Note AP-155, "Oscillators for Microcontrollers," available from your Intel sale representative.

To drive the device from an external clock source, XTAL1 should be driven, while XTAL2 is left unconnected, as shown in Figure 4. There are no requirements on the duty cycle of the external clock signal, since the input to the internal clocking circuitry is through a divide-by-two flip-flop, but minimum and maximum high and low times specified on the Data Sheet must be observed.

## DESIGN CONSIDERATIONS

- At power on, the voltage on  $V_{CC}$  and RST must come up at the same time for a proper start-up.
- Before entering the Power Down mode the contents of the Carry Bit and B. 7 must be equal.

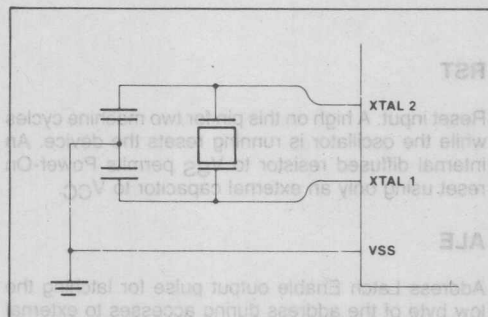


Figure 3. Crystal Oscillator

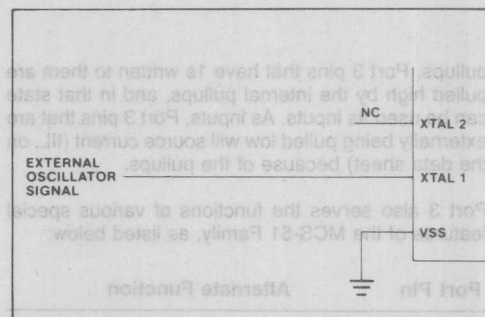


Figure 4. External Drive Configuration



# **ABSOLUTE MAXIMUM RATINGS\***

Ambient Temperature Under Bias . . . 0°C to 70°C

Storage Temperature . . . . . -65°C to +150°C

Voltage on Any Pin to V<sub>SS</sub> . . . . . -0.5 V to V<sub>CC</sub> + 0.5 V

Voltage on V<sub>CC</sub> to V<sub>SS</sub> . . . . . -0.5 V to 6.5 V

Power Dissipation . . . . . 1.0 W\*\*

\*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

## **D.C. CHARACTERISTICS:** (T<sub>A</sub> = 0°C to 70°C; V<sub>CC</sub> = 5 V ± 20%; V<sub>SS</sub> = 0 V)

| Symbol           | Parameter  | Min                    | Max                    | Unit | Test Conditions                                       |
|------------------|--|------------------------|------------------------|------|---|
| V <sub>IL</sub>  | Input Low Voltage (Except EA)                                | -0.5                   | .2V <sub>CC</sub> - .1 | V    |   |
| V <sub>IL1</sub> | Input Low Voltage (EA)                                       | -0.5                   | .2V <sub>CC</sub> - .3 | V    |   |
| V <sub>IH</sub>  | Input High Voltage (Except XTAL1, RST)                       | .2V <sub>CC</sub> + .9 | V <sub>CC</sub> + 0.5  | V    |   |
| V <sub>IH1</sub> | Input High Voltage (XTAL1, RST)                              | .7V <sub>CC</sub>      | V <sub>CC</sub> + 0.5  | V    |   |
| V <sub>OL</sub>  | Output Low Voltage (Ports 1, 2, 3)                           |                        | 0.45                   | V    | I <sub>OL</sub> = 1.6 mA (Note 1)                     |
| V <sub>OL1</sub> | Output Low Voltage (Port 0, ALE, PSEN)                       |                        | 0.45                   | V    | I <sub>OL</sub> = 3.2 mA (Note 1)                     |
| V <sub>OH</sub>  | Output High Voltage (Ports 1, 2, 3)                          | 2.4                    |                        | V    | I <sub>OH</sub> = -60 μA V <sub>CC</sub> = 5 V ± 10%  |
|                  |  | .75V <sub>CC</sub>     |                        | V    | I <sub>OH</sub> = -25 μA                              |
|                  |  | .9V <sub>CC</sub>      |                        | V    | I <sub>OH</sub> = -10 μA                              |
| V <sub>OH1</sub> | Output High Voltage (Port 0 in External Bus Mode, ALE, PSEN) | 2.4                    |                        | V    | I <sub>OH</sub> = -400 μA V <sub>CC</sub> = 5 V ± 10% |
|                  |  | .75V <sub>CC</sub>     |                        | V    | I <sub>OH</sub> = -150 μA                             |
|                  |  | .9V <sub>CC</sub>      |                        | V    | I <sub>OH</sub> = -40 μA (Note 2)                     |
| I <sub>IL</sub>  | Logical 0 Input Current (Ports 1, 2, 3)                      |                        | -50                    | μA   | V <sub>in</sub> = 0.45 V                              |
| I <sub>TL</sub>  | Logical 1 to 0 Transition Current (Ports 1, 2, 3)            |                        | -650                   | μA   | V <sub>in</sub> = 2 V                                 |
| I <sub>LI</sub>  | Input Leakage Current (Port 0, EA)                           |                        | ± 10                   | μA   | 0.45 < V <sub>in</sub> < V <sub>CC</sub>              |
| RRST             | Reset Pulldown Resistor                                      | 50                     | 150                    | KOhm |   |
| CIO              | Pin Capacitance  |                        | 10                     | pF   | Test Freq = 1 MHz, T <sub>A</sub> = 25°C              |
| IPD              | Power Down Current   |                        | 50                     | μA   | V <sub>CC</sub> = 2 to 6 V (Note 3)                   |

\*\*This value is based on the maximum allowable die temperature and the thermal resistance of the package.

**TYPICAL ICC:** Typical operating ICC at 3 MHz and  $V_{CC} = 5.0$  V is 3.8 mA, and varies with frequency at the rate of 0.67 mA/MHz.

**MAXIMUM ICC (mA)**

| Freq. $V_{CC}$ | Operating (Note 4) |      |     | Idle (Note 5) |     |     |
|----------------|--------------------|------|-----|---------------|-----|-----|
|                | 4 V                | 5 V  | 6 V | 4 V           | 5 V | 6 V |
| 0.5 MHz        | 1.6                | 2.2  | 3   | 0.6           | 0.9 | 1.2 |
| 3.5 MHz        | 4.3                | 5.7  | 7.5 | 1.1           | 1.6 | 2.2 |
| 8.0 MHz        | 8.3                | 11   | 14  | 1.8           | 2.7 | 3.7 |
| 12 MHz         | 12                 | 16   | 20  | 2.5           | 3.7 | 5   |
| 16 MHz         | 16                 | 20.5 | 25  | 3.5           | 5   | 6.5 |

**Note 1:** Capacitive loading on Ports 0 and 2 may cause spurious noise pulses to be superimposed on the  $V_{OLS}$  of ALE and Ports 1 and 3. The noise is due to external bus capacitance discharging into the Port 0 and Port 2 pins when these pins make 1-to-0 transitions during bus operations. In the worst cases (capacitive loading  $> 100$  pF), the noise pulse on the ALE line may exceed 0.8 V. In such cases it may be desirable to qualify ALE with a Schmitt Trigger, or use an address latch with a Schmitt Trigger STROBE input.

**Note 2:** Capacitive loading on Ports 0 and 2 may cause the  $V_{OH}$  on ALE and PSEN to momentarily fall before the .9  $V_{CC}$  specification when the address bits are stabilizing.

**Note 3:** Power Down  $ICC$  is measured with all output pins disconnected; EA = Port 0 =  $V_{CC}$ ; XTAL2 N.C.; RST =  $V_{SS}$ .

**Note 4:**  $ICC$  is measured with all output pins disconnected; XTAL1 driven with TCLCH, TCHCL = 5 ns,  $V_{IL} = V_{SS} + .5$  V,  $V_{IH} = V_{CC} - .5$  V; XTAL2 N.C.; EA = RST = Port 0 =  $V_{CC}$ .  $ICC$  would be slightly higher if a crystal oscillator is used.

**Note 5:** Idle  $ICC$  is measured with all output pins disconnected; XTAL1 driven with TCLCH, TCHCL = 5 ns,  $V_{IL} = V_{SS} + .5$  V,  $V_{IH} = V_{CC} - .5$  V; XTAL2 N.C.; Port 0 =  $V_{CC}$ ; EA = RST =  $V_{SS}$ .

**EXPLANATION OF THE AC SYMBOLS**

Each timing symbol has 5 characters. The first character is always a 'T' (stands for time). The other characters, depending on their positions, stand for the name of a signal or the logical status of that signal. The following is a list of all the characters and what they stand for.

- A: Address.
- C: Clock.
- D: Input data.
- H: Logic level HIGH.
- I: Instruction (program memory contents).
- L: Logic level LOW, or ALE.

- P: PSEN.
- Q: Output data.
- R: READ signal.
- T: Time.
- V: Valid.
- W: WRITE signal.
- X: No longer a valid logic level.
- Z: Float.

**EXAMPLE:**

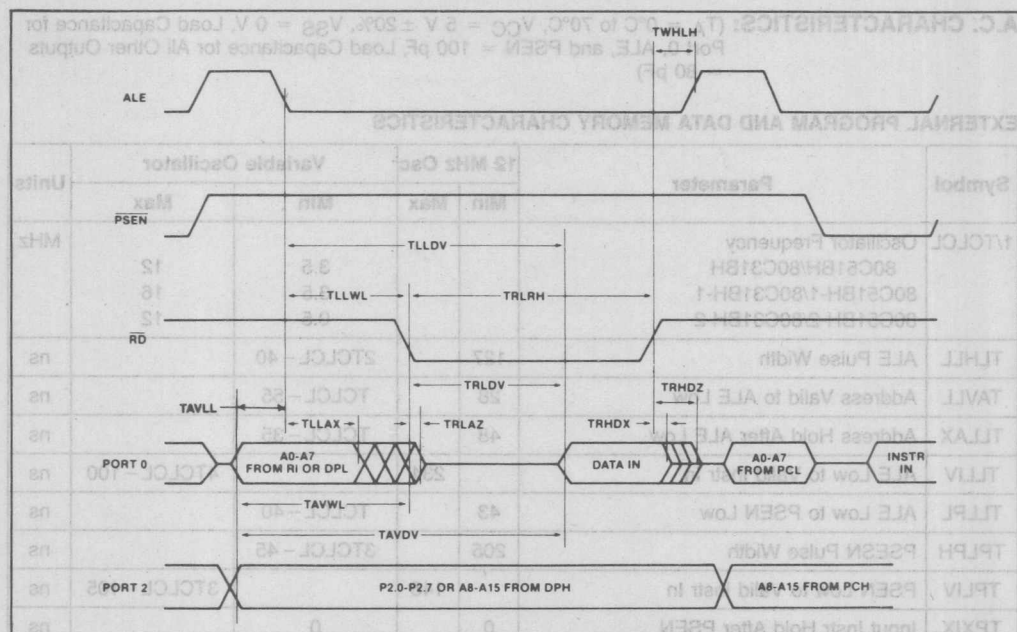
- TAVLL = Time for Address Valid to ALE Low.
- TLLPL = Time for ALE Low to PSEN Low.

# 80C51BH/80C51BH-1/80C51BH-2

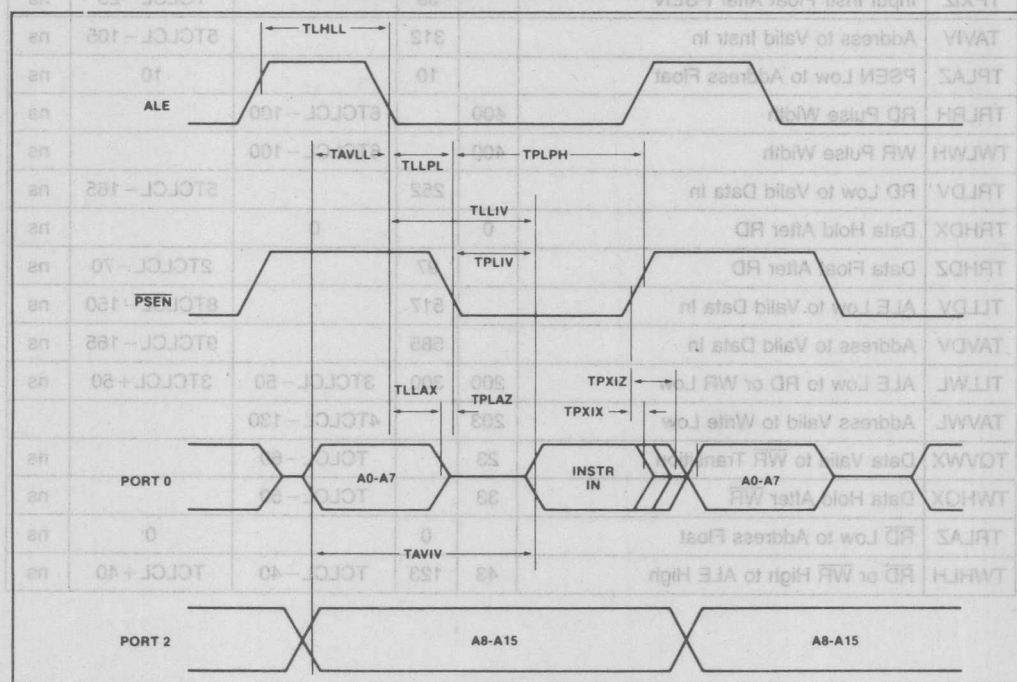
Port 0, ALE, and PSEN = 100 pF, Load Capacitance for All Other Outputs = 80 pF)

## EXTERNAL PROGRAM AND DATA MEMORY CHARACTERISTICS

| Symbol  | Parameter   | 12 MHz Osc |     | Variable Oscillator |                | Units |
|---------|---|------------|-----|---------------------|----------------|-------|
|         |   | Min        | Max | Min                 | Max            |       |
| 1/TCLCL | Oscillator Frequency<br>80C51BH/80C31BH<br>80C51BH-1/80C31BH-1<br>80C51BH-2/80C31BH-2 |            |     | 3.5<br>3.5<br>0.5   | 12<br>16<br>12 | MHz   |
| TLHLL   | ALE Pulse Width   | 127        |     | 2TCLCL - 40         |                | ns    |
| TAVLL   | Address Valid to ALE Low  | 28         |     | TCLCL - 55          |                | ns    |
| TLLAX   | Address Hold After ALE Low  | 48         |     | TCLCL - 35          |                | ns    |
| TLLIV   | ALE Low to Valid Instr In   |            | 234 |                     | 4TCLCL - 100   | ns    |
| TLLPL   | ALE Low to PSEN Low   | 43         |     | TCLCL - 40          |                | ns    |
| TPLPH   | PSEN Pulse Width  | 205        |     | 3TCLCL - 45         |                | ns    |
| TPLIV   | PSEN Low to Valid Instr In  |            | 145 |                     | 3TCLCL - 105   | ns    |
| TPXIX   | Input Instr Hold After PSEN   | 0          |     | 0                   |                | ns    |
| TPXIZ   | Input Instr Float After PSEN  |            | 59  |                     | TCLCL - 25     | ns    |
| TAVIV   | Address to Valid Instr In   |            | 312 |                     | 5TCLCL - 105   | ns    |
| TPLAZ   | PSEN Low to Address Float   |            | 10  |                     | 10             | ns    |
| TRLRH   | RD Pulse Width  | 400        |     | 6TCLCL - 100        |                | ns    |
| TWLWH   | WR Pulse Width  | 400        |     | 6TCLCL - 100        |                | ns    |
| TRLDV   | RD Low to Valid Data In   |            | 252 |                     | 5TCLCL - 165   | ns    |
| TRHDX   | Data Hold After RD  | 0          |     | 0                   |                | ns    |
| TRHDZ   | Data Float After RD   |            | 97  |                     | 2TCLCL - 70    | ns    |
| TLLDV   | ALE Low to Valid Data In  |            | 517 |                     | 8TCLCL - 150   | ns    |
| TAVDV   | Address to Valid Data In  |            | 585 |                     | 9TCLCL - 165   | ns    |
| TLLWL   | ALE Low to RD or WR Low   | 200        | 300 | 3TCLCL - 50         | 3TCLCL + 50    | ns    |
| TAVWL   | Address Valid to Write Low  | 203        |     | 4TCLCL - 130        |                |       |
| TQVWX   | Data Valid to $\overline{WR}$ Transition  | 23         |     | TCLCL - 60          |                | ns    |
| TWHQX   | Data Hold After $\overline{WR}$   | 33         |     | TCLCL - 50          |                | ns    |
| TRLAZ   | $\overline{RD}$ Low to Address Float  |            | 0   |                     | 0              | ns    |
| TWHLH   | $\overline{RD}$ or $\overline{WR}$ High to ALE High                                   | 43         | 123 | TCLCL - 40          | TCLCL + 40     | ns    |

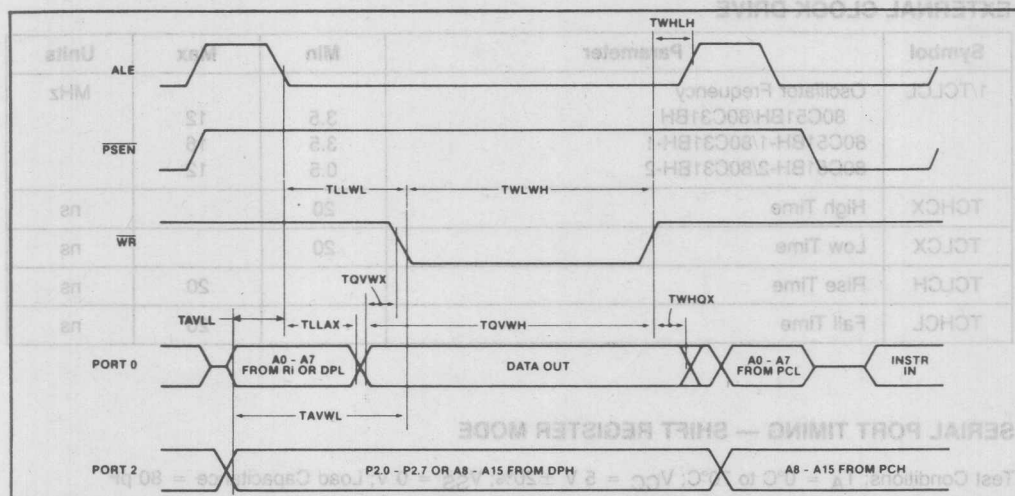


### EXTERNAL DATA MEMORY READ CYCLE

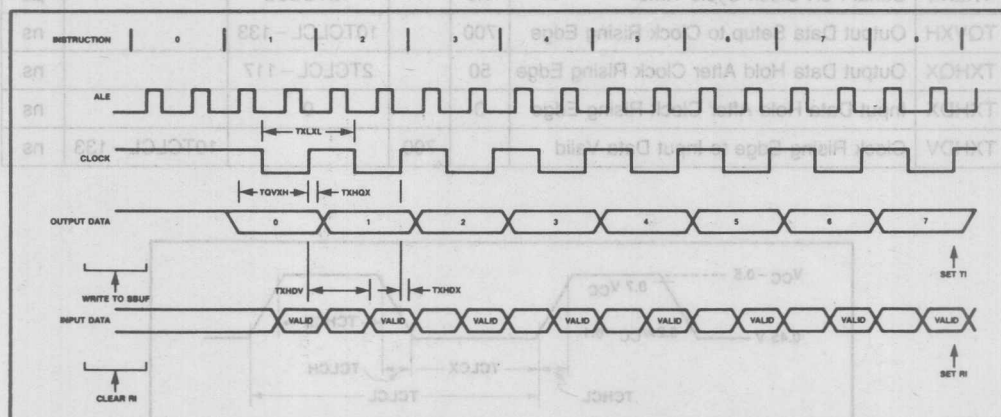


### EXTERNAL PROGRAM MEMORY READ CYCLE





### EXTERNAL DATA MEMORY WRITE CYCLE



### SHIFT REGISTER MODE TIMING WAVEFORMS



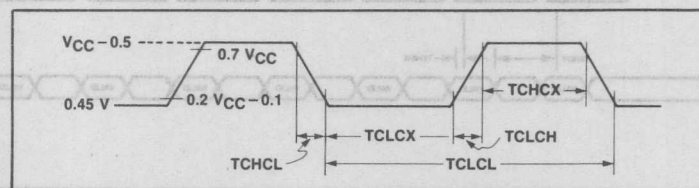
## EXTERNAL CLOCK DRIVE

| Symbol  | Parameter            | Min | Max | Units |
|---------|----------------------|-----|-----|-------|
| 1/TCLCL | Oscillator Frequency |     |     | MHz   |
|         | 80C51BH/80C31BH      | 3.5 | 12  |       |
|         | 80C51BH-1/80C31BH-1  | 3.5 | 16  |       |
|         | 80C51BH-2/80C31BH-2  | 0.5 | 12  |       |
| TCHCX   | High Time            | 20  |     | ns    |
| TCLCX   | Low Time             | 20  |     | ns    |
| TCLCH   | Rise Time            |     | 20  | ns    |
| TCHCL   | Fall Time            |     | 20  | ns    |

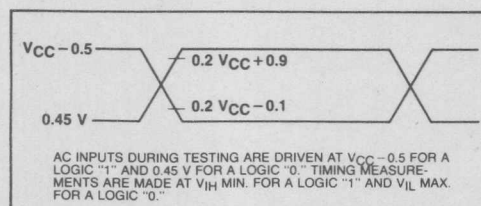
## SERIAL PORT TIMING — SHIFT REGISTER MODE

Test Conditions:  $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ;  $V_{CC} = 5\text{ V} \pm 20\%$ ;  $V_{SS} = 0\text{ V}$ ; Load Capacitance =  $80\text{ pF}$

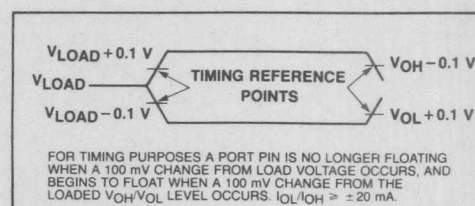
| Symbol | Parameter                                | 12 MHz Osc |     | Variable Oscillator |               | Units         |
|--------|--|------------|-----|---------------------|---------------|---------------|
|        |  | Min        | Max | Min                 | Max           |               |
| TXLXL  | Serial Port Clock Cycle Time             | 1.0        |     | 12TCLCL             |               | $\mu\text{s}$ |
| TQVXH  | Output Data Setup to Clock Rising Edge   | 700        |     | 10TCLCL - 133       |               | ns            |
| TXHQX  | Output Data Hold After Clock Rising Edge | 50         |     | 2TCLCL - 117        |               | ns            |
| TXHDX  | Input Data Hold After Clock Rising Edge  | 0          |     | 0                   |               | ns            |
| TXHDV  | Clock Rising Edge to Input Data Valid    |            | 700 |                     | 10TCLCL - 133 | ns            |



EXTERNAL CLOCK DRIVE WAVEFORM



AC TESTING INPUT, OUTPUT WAVEFORMS



FLOAT WAVEFORMS

## CHMOS SINGLE COMPONENT 8-BIT MICROCONTROLLER with Factory Mask-Programmable ROM

### CHMOS SINGLE COMPONENT 8-BIT CONTROL-ORIENTED CPU with RAM and I/O

80C51BH/80C31BH — 3.5 to 12 MHz  $V_{CC} = 5V \pm 20\%$

- Extended Temperature Range
- Burn-In

The Intel EXPRESS system offers enhancements to the operational specifications of the MCS-51 family of microcontrollers. These EXPRESS products are designed to meet the needs of those applications whose operating requirements exceed commercial standards.

The EXPRESS program includes the commercial standard temperature range with burn-in, and an extended temperature range with or without burn-in.

With the commercial standard temperature range operational characteristics are guaranteed over the temperature range of 0°C to 70°C. With the extended temperature range option, operational characteristics are guaranteed over the range of -40°C to +85°C.

The optional burn-in is dynamic, for a minimum time of 160 hours at 125°C with  $V_{CC} = 6.0V \pm 0.25V$ , following guidelines in MIL-STD-883, Method 1015.

Package types and EXPRESS versions are identified by a one- or two-letter prefix to the part number. The prefixes are listed in Table 1.

For the extended temperature range option, this data sheet specifies the parameters which deviate from their commercial temperature range limits. The commercial temperature range data sheets are applicable for all parameters not listed here.

### Electrical Deviations from Commercial Specifications for Extended Temperature Range

D.C. and A.C. parameters not included here are the same  
as in the commercial temperature range data sheets.

**D.C. CHARACTERISTICS:** ( $T_A = -40^\circ\text{C}$  to  $+85^\circ\text{C}$ ;  $V_{CC} = 5\text{V} \pm 20\%$ ;  $V_{SS} = 0\text{V}$ )

| Symbol    | Parameter  | Limits            |                  | Unit          | Test Conditions         |
|-----------|--|-------------------|------------------|---------------|-------------------------|
|           |  | Min.              | Max.             |               |                         |
| $V_{IL}$  | Input Low Voltage<br>(Except EA)                     | -0.5              | $.2V_{CC} - .15$ | V             |                         |
| $V_{IL1}$ | EA   | -0.5V             | $.2V_{CC} - .35$ | V             |                         |
| $V_{IH}$  | Input High Voltage<br>(Except XTAL1, RST)            | $.2V_{CC} + 1$    | $V_{CC} + 0.5$   | V             |                         |
| $V_{IH1}$ | Input High Voltage to<br>XTAL1, RST                  | $0.7V_{CC} + 0.1$ | $V_{CC} + 0.5$   | V             |                         |
| $I_{IL}$  | Logical 0 Input Current<br>(Port 1, 2, 3)            | -75               |                  | $\mu\text{A}$ | $V_{in} = 0.45\text{V}$ |
| $I_{TL}$  | Logical 1 to 0 transition<br>Current (Ports 1, 2, 3) | -750              |                  | $\mu\text{A}$ | $V_{in} = 2.0\text{V}$  |

Table 1 — Prefix Identification

| Prefix | Package Type | Temperature Range | Burn-In |
|--------|--------------|-------------------|---------|
| P      | Plastic      | Commercial        | No      |
| D      | Cerdip       | Commercial        | No      |
| N      | PLCC         | Commercial        | No      |
| TP     | Plastic      | Extended          | No      |
| TD     | Cerdip       | Extended          | No      |
| TN     | PLCC         | Extended          | No      |
| QP     | Plastic      | Commercial        | Yes     |
| QD     | Cerdip       | Commercial        | Yes     |
| QN     | PLCC         | Commercial        | Yes     |
| LP     | Plastic      | Extended          | Yes     |
| LD     | Cerdip       | Extended          | Yes     |
| LN     | PLCC         | Extended          | Yes     |

**Please Note:**

- Commercial temperature range is 0° to 70°C. Extended temperature range is -40° to +85°C.
- Burn-in is dynamic, for a minimum time of 160 hours at 125°C,  $V_{CC} = 6.0 \pm 0.25V$ , following guidelines in MIL-STD-883 Method 1015 (Test Condition D).

**Examples:**

P80C31BH indicates 80C31BH in a plastic package and specified for commercial temperature range, without burn-in.

LD80C51BH indicates 80C51BH in a cerdip package and specified for extended temperature range with burn-in.

D.C. CHARACTERISTICS:  $T_A = -40^\circ C$  to  $+85^\circ C$ ;  $V_{CC} = 5V \pm 10\%$ ;  $V_{SS} = 0V$

| Symbol    | Parameter                              | Min  | Max            | Unit | Test Conditions                          |
|-----------|--|------|----------------|------|--|
| $V_{IL}$  | Input Low Voltage                      | -0.5 | 0.75           | V    |  |
| $V_{IH}$  | Input High Voltage (except XTAL2, RST) | 2.1  | $V_{CC} + 0.5$ | V    |  |
| $I_{CC}$  | Power Supply Current                   |      | 135            | mA   | All Outputs Disconnected; $E_A = V_{CC}$ |
|           |  |      | 175            | mA   |  |
|           |  |      | 285            | mA   |  |
| $I_{IL2}$ | Logic 0 Input Current (XTAL2)          | -4.0 |                | mA   | $V_{IL} = 0.45V$                         |

**8031AH/8051AH  
8032AH/8052AH  
8751H/8751H**

**EXPRESS**

| Prefix | Package Type | Temperature Range          | Burn-in |
|--------|--------------|----------------------------|---------|
| P      | Plastic      | Commercial                 | No      |
| D      | CerDip       | Commercial                 | No      |
| N      | PLCC         | Commercial                 | No      |
| TP     | Plastic      | Extended Temperature Range | No      |
| TD     | CerDip       | Extended Temperature Range | No      |
| TN     | PLCC         | Extended Temperature Range | No      |
| CP     | CerDip       | Commercial                 | No      |
| LP     | Plastic      | Extended                   | Yes     |
| LN     | PLCC         | Extended                   | Yes     |

The Intel EXPRESS system offers enhancements to the operational specifications of the MCS®-51 family of microcontrollers. These EXPRESS products are designed to meet the needs of those applications whose operating requirements exceed commercial standards.

The EXPRESS program includes the commercial standard temperature range with burn-in, and an extended temperature range with or without burn-in.

With the commercial standard temperature range operational characteristics are guaranteed over the temperature range of 0°C to 70°C. With the extended temperature range option, operational characteristics are guaranteed over the range of -40°C to +85°C.

The optional burn-in is dynamic, for a minimum time of 160 hours at 125°C with  $V_{CC} = 5.5V \pm 0.5V$ , following guidelines in MIL-STD-883, Method 1015.

Package types and EXPRESS versions are identified by a one- or two-letter prefix to the part number. The prefixes are listed in Table 1.

For the extended temperature range option, this data sheet specifies the parameters which deviate from their commercial temperature range limits. The commercial temperature range data sheets are applicable for all parameters not listed here.

**Electrical Deviations from Commercial Specifications for Extended Temperature Range**

D.C. and A.C. parameters not included here are the same as in the commercial temperature range data sheets.

**D.C. CHARACTERISTICS:** ( $T_A = -40^\circ\text{C}$  to  $+85^\circ\text{C}$ ;  $V_{CC} = 5V \pm 10\%$ ;  $V_{SS} = 0V$ )

| Symbol    | Parameter  | Min  | Max               | Unit           | Test Conditions                                |
|-----------|--|------|-------------------|----------------|--|
| $V_{IL}$  | Input Low Voltage  | -0.5 | 0.75              | V              |  |
| $V_{IH}$  | Input High Voltage (Except XTAL2, RST)                                   | 2.1  | $V_{CC} + 0.5$    | V              |  |
| $I_{CC}$  | Power Supply Current:<br>8051AH,8031AH<br>8052AH,8032AH<br>8751H,8751H-8 |      | 135<br>175<br>265 | ma<br>ma<br>ma | All Outputs<br>Disconnected;<br>$E_A = V_{CC}$ |
| $I_{IL2}$ | Logic 0 Input Current (XTAL2)  |      | -4.0              | ma             | $V_{in} = 0.45\text{ V}$                       |



Table 1 — Prefix Identification

| Prefix | Package Type | Temperature Range | Burn-In |
|--------|--------------|-------------------|---------|
| P      | plastic      | commercial        | no      |
| D      | cerdip       | commercial        | no      |
| C      | ceramic      | commercial        | no      |
| TP     | plastic      | extended          | no      |
| TD     | cerdip       | extended          | no      |
| TC     | ceramic      | extended          | no      |
| QP     | plastic      | commercial        | yes     |
| QD     | cerdip       | commercial        | yes     |
| QC     | ceramic      | commercial        | yes     |
| LP     | plastic      | extended          | yes     |
| LD     | cerdip       | extended          | yes     |
| LC     | ceramic      | extended          | yes     |

Please note:

- Commercial temperature range is 0° to 70°C. Extended temperature range is -40° to +85°C.
- Burn-in is dynamic, for a minimum time of 160 hours at 125°C,  $V_{CC} = 5.5V \pm 0.5V$ , following guidelines in MIL-STD-883 Method 1015 (Test Condition D).
- The following devices are not available in plastic packages:  
8751H, 8751H
- The following devices are not available in ceramic packages:  
8051AH, 8031AH  
8052AH, 8032AH

Examples: P8031AH indicates 8031AH in a plastic package and specified for commercial temperature range, without burn-in. LD8751H indicates 8751H in a cerdip package and specified for extended temperature range with burn-in.

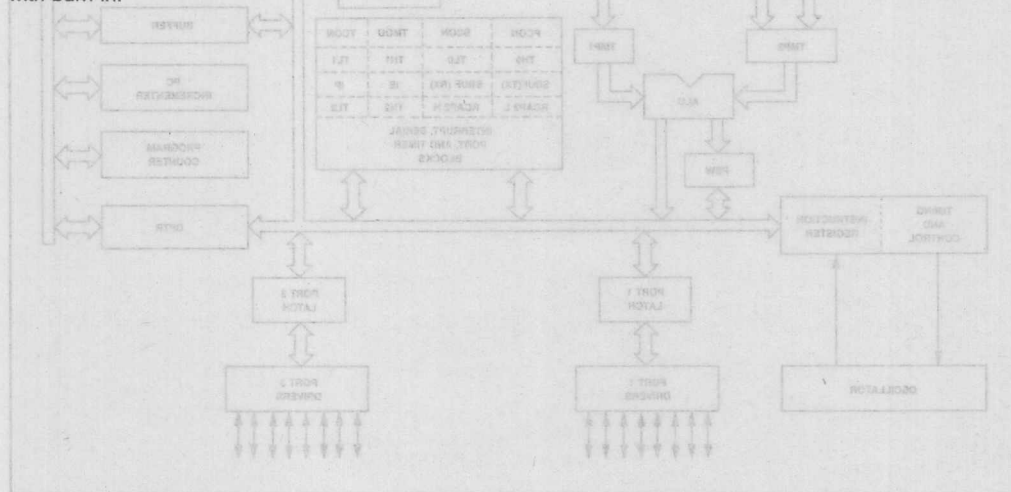


Figure 1. Block Diagram

# 8752A

## SINGLE-CHIP 8-BIT MICROCONTROLLER

### THREE 16-BIT TIMER/COUNTERS

### 8K BYTES OF EPROM PLUS 256 BYTES OF RAM

- 2-Level Program Security System
- 8K Bytes EPROM
- 256 Bytes Data RAM
- intelligent Programming™ Algorithm
- 12.5 V Programming Voltage
- Boolean Processor
- 32 Programmable I/O Lines
- Three 16-Bit Timer/Counters
- 6 Interrupt Sources
- Programmable Serial Channel
- Separate Transmit/Receive Baud Rate Capability
- 64K Program Memory Space
- 64K Data Memory Space
- LCC and DIP packagings available

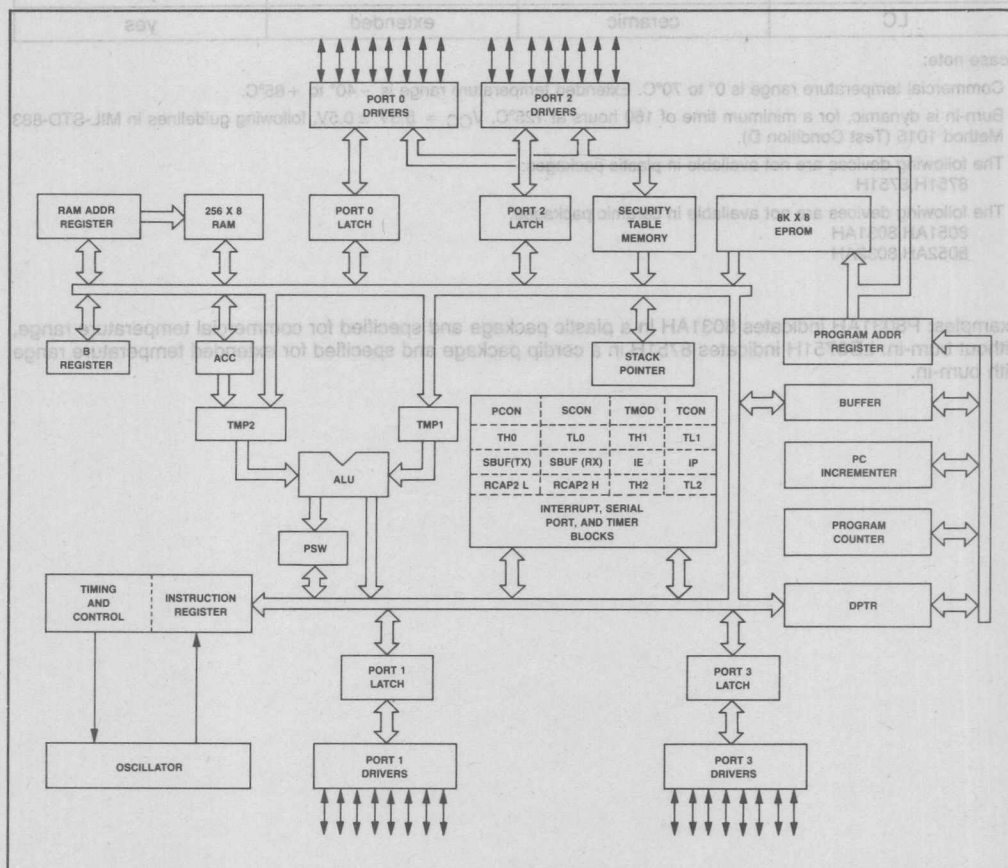


Figure 1. Block Diagram

The Intel 8752A is the EPROM version of the 8052AH. It contains 8K x 8 of on-chip Program memory that can be electrically programmed, and can be erased by exposure to ultraviolet light.

The 8752A is a member of the MCS®-51 microcontroller family that are optimized for control applications. Byte processing and numerical operation on small data structures are facilitated by a variety of fast addressing modes for accessing the internal RAM. The instruction set provides a convenient menu of 8-bit arithmetic instructions, including a 4 micro-second (@ 12 MHz) multiply and divide instructions.

The 8752A also offers two new features; a 2-level security memory system and intelligent programming™ algorithm.

The two-level program security system consists of 2 security bits and a 32 byte security table memory which are used to protect the program memory against software piracy.

The intelligent programming™ algorithm reduces the programming time from 50 msec per byte to a minimum of 4 msec per byte.

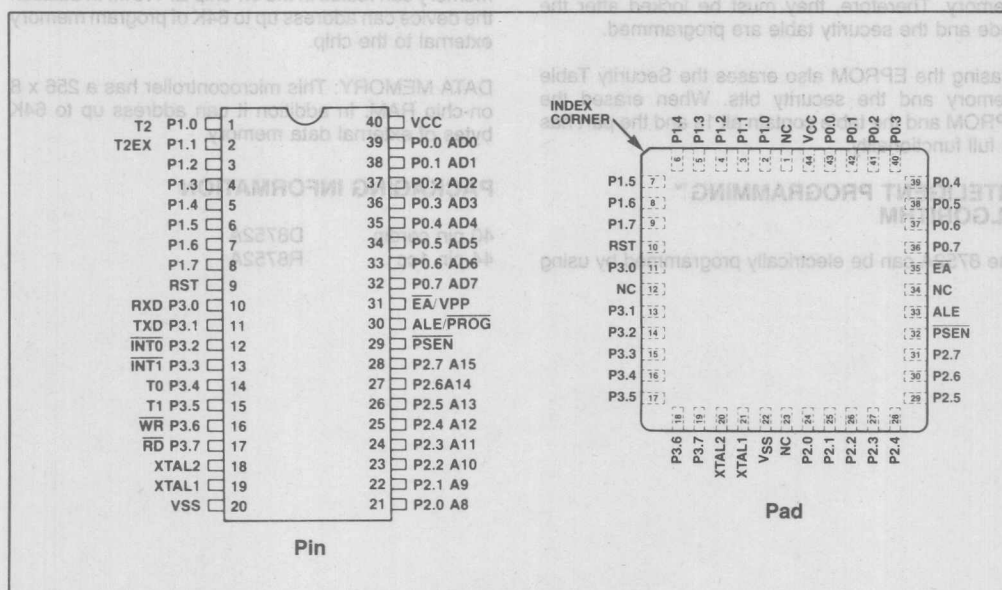


Figure 2. Pin Connection

## THE NEW TWO LEVEL PROGRAM SECURITY SYSTEM

The security system of the 8752A is designed to give the user the maximum control in protecting the internal program memory of the part. It allows the user to apply the degree of security suitable for the application.

Two security bits are implemented in the 8752A; the

SECURE EXTERNAL EXECUTION, and the VERIFY bit. Programming both bits denies any external access to the on-chip program memory.

The security bits, when programmed, prohibit the controller from reading or moving the internal code when executing out of external program memory, and also disable the verify mode.

It is possible to maintain the verify capability while securing the code. This is done by programming only one bit (SECURE EXTERNAL EXECUTION), and programming the SECURITY TABLE. The data that appears on the port during the secured verification is scrambled BY the SECURITY TABLE contents.

The SECURITY TABLE is a 32 x 8 array of EPROM. The user can electrically program any arbitrary code into it. The code will be used to mask the program memory during the verification. Thus what appears at port 0 during the verification is a scrambled code which cannot be deciphered without the key table.

Programming the security bits also denies the programming of the EPROM and the Security Table Memory. Therefore, they must be locked after the code and the security table are programmed.

Erasing the EPROM also erases the Security Table Memory and the security bits. When erased the EPROM and the table contain all 1s and the part has its full functionality.

### INTELLIGENT PROGRAMMING™ ALGORITHM

The 8752A can be electrically programmed by using

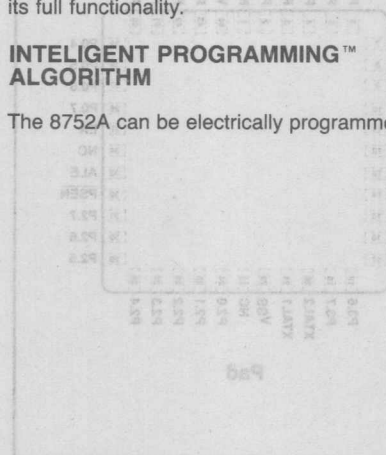


Figure 2. Pin Connection

the intelligent programming algorithm. This method is faster and more efficient than the conventional programming method. This process, instead of programming each byte for 50 msec, tries 1 msec per byte and verifies it. Normally that is enough to burn the data in, in which case a final-programming pulse is applied. If the correct data cannot be verified, the programming is repeated, up to a maximum of 15 times, followed by the 3X final-programming signal. X is the number of times it took to verify the correct data.

### MEMORY ORGANIZATION

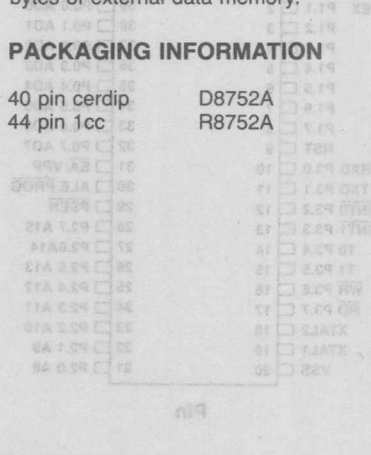
**PROGRAM MEMORY:** Up to 8K bytes of the program memory can reside in the on-chip EPROM. In addition the device can address up to 64K of program memory external to the chip.

**DATA MEMORY:** This microcontroller has a 256 x 8 on-chip RAM. In addition it can address up to 64K bytes of external data memory.

### PACKAGING INFORMATION

40 pin cerdip  
44 pin 1cc

D8752A  
R8752A



### THE NEW TWO LEVEL PROGRAM SECURITY SYSTEM

The security system of the 8752A is designed to give the user the maximum control in protecting the internal program memory of the part. It allows the user to apply the degree of security suitable for the application.

Two security bits are implemented in the 8752A; the

SECURE EXTERNAL EXECUTION, and the VERIFY. Programming both bits denies any external access to the on-chip program memory. The security bits, when programmed, prohibit the controller from reading or moving the internal code when executing out of external program memory, and also disable the verify mode.



# **CHMOS SINGLE-CHIP 8-BIT MICROCONTROLLER** **HSO, PWM, COMPARATOR/CAPTURE,** **UP/DOWN COUNTER**

**80C252 — CPU WITH RAM AND I/O**  
**83C252 — 8K BYTES FACTORY MASK**  
**PROGRAMMABLE ROM**

**87C252 — 8K BYTES USER PROGRAMMABLE EPROM**

- High Performance CHMOS Process
- Power Control Modes
- Programmable Counter Array
- High Speed OUTPUT
- Pulse Width Modulator
- Up/Down Timer/Counter
- Watchdog Timer
- Two Level Program Security System
- 8K Factory Mask ROM
- 256 Bytes of On-Chip Data RAM
- intelligent Programming™ Algorithm
- Boolean Processor
- 32 Programmable I/O Lines
- Three 16-Bit Timer/Counters
- 7 Interrupt Sources
- Programmable Serial Channel
- Framing Error Detection
- Automatic Address Recognition
- TTL Compatible Logic Levels
- 64K Program Memory Space
- 64K Data Memory Space
- MCS®-51 Fully Compatible Inst. Set
- PLCC and DIP packagings

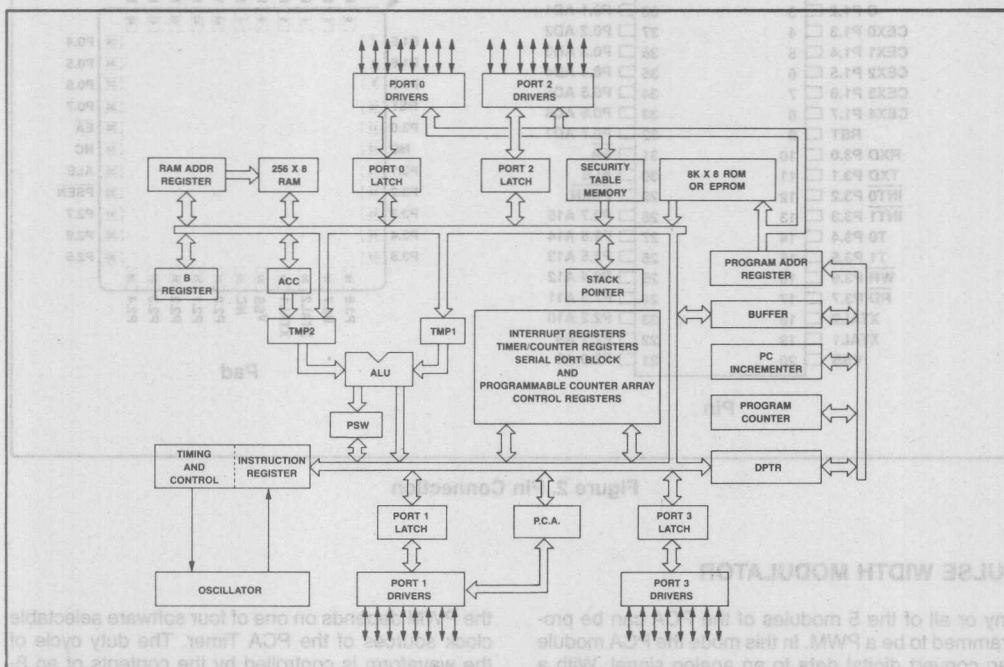


Figure 1. Block Diagram



The Intel 80C252 is a single-chip control oriented microcontroller which is fabricated on Intel's reliable CHMOS-III technology. Being a member of the MCS-51 family, the 80C252 uses the same powerful instruction set, has the same architecture, and is pin for pin backward compatible with the existing MCS-51 products.

The 80C252 has several new features that make it even more powerful than the 80C51BH. These features are: an additional 128 x 8 bytes of on-chip RAM, an extra Timer/Counter with up/down counting capability, a Programmable Counter Array, and Framing Error Detection and Automatic Address Recognition for the Serial Port.

The Programmable Counter Array consists of five modules and five I/O pins (share Port 1 pins). Each module is capable of being programmed as: a Pulse Width Modulator, a Watchdog Timer, a Compare/Capture Register, a High Speed Output, or just a plain 16-bit Timer/Counter. The PCA can generate one interrupt when programmed as a Timer or as a Compare/Capture register.

With the Pulse Width Modulator and the High Speed Output capability, the 80C252 can be used in a wide range of motor control applications, and the Serial Port enhancements provide extra reliability in the multi-processor serial communication environments.

The overall power consumption of this part, plus the two power control modes (Idle and Power Down), make this part ideal for low power and/or battery operated applications.

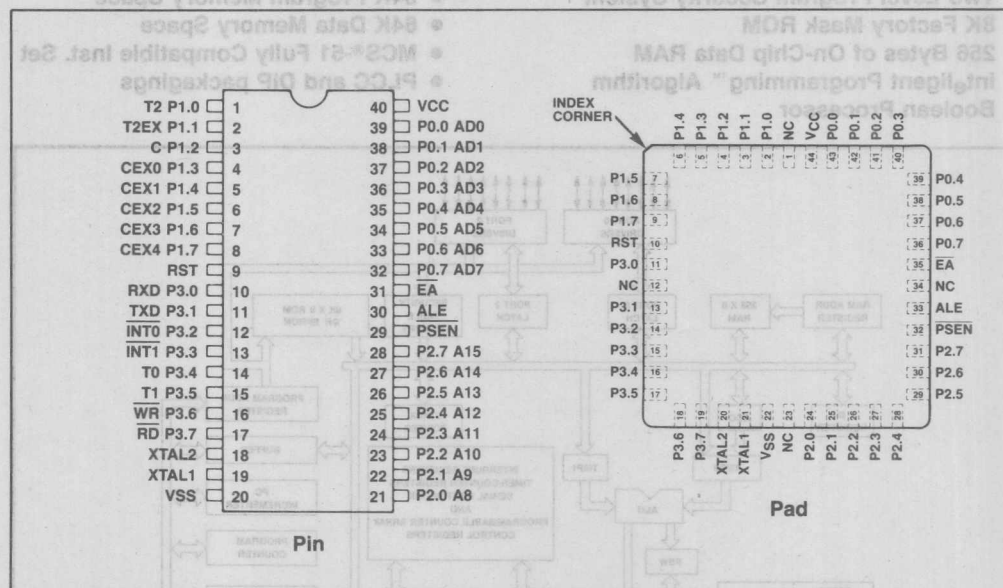


Figure 2. Pin Connection

## PULSE WIDTH MODULATOR

Any or all of the 5 modules of the PCA can be programmed to be a PWM. In this mode the PCA module can convert digital data to an analog signal. With a 16 MHz clock the maximum frequency of the output waveform of the PWM is 15.6 KHz. The frequency of

the PWM depends on one of four software selectable clock sources of the PCA Timer. The duty cycle of the waveform is controlled by the contents of an 8-bit register that can be programmed to be any integer from 0 to 255.

### HIGH SPEED OUTPUT

Any of the PCA modules can be programmed to generate a signal which appears on the corresponding I/O pin. The output frequency can be programmed to a maximum of 6.5 KHz.

### COMPARATOR/CAPTURE

When programmed as a comparator, the PCA module, at every cycle, compares the contents of its Timer with the preset value of the Compare register. When a match occurs, it reverses the logic level of its corresponding I/O pin, and generates an interrupt.

In the Capture mode the process is the reverse of the Comparator. Upon changing the logic level on the corresponding I/O pin of the module, the content of the PCA Timer is loaded into the selected Capture register. An interrupt may also be generated.

These two features allow the 80C151 to be used in accurate pulse width measurement in real time, with minimum software overhead.

### WATCHDOG TIMER

Module number 4 of the PCA, along with its Timer can be programmed to play Watchdog Timer. If the Timer value matches the Comparator Register value, an internal Reset signal is generated which puts the microcontroller in a hardware reset.

During normal operation the software, periodically, reloads the Timer or disables the internal reset signal. Therefore, in case of a malfunction due to noise or any unwanted situation, the processor recovers itself and provides a more reliable operation.

### TIMER/COUNTER

Timer 2 is a 16-bit timer/counter which is capable of up or down counting. It has a 16-bit register that can be used for auto-reloading or as a capture register. The capture command can come from external source as well as the clock input to the timer. With the help of Timer 2, two different baud rates can be generated for transmit and receive. Timer 2 can be utilized while the processor is in the Idle Mode.

### SERIAL PORT

The full duplex serial port of the 80C252 is the same as the serial port of the 80C51 with two new features; Framing Error Detection, and Automatic Address Recognition.

When the Serial Port is operating in either mode 2 or

3, and the Framing Error Detection system is enabled, an invalid received byte affects a status bit. By checking the bit in the software, immediately after each reception, one can distinguish between correct and incorrect bytes.

The 80C252, as a slave microcontroller, can recognize when it is being addressed by a master or another slave controller in a multi-processor environment.

### THE NEW TWO-LEVEL PROGRAM SECURITY SYSTEM

The security system of the 87C252 and the 83C252 is designed to give the user the maximum control in protecting the internal program memory of the part. It allows the user to apply the degree of security suitable for the application.

Two security bits are implemented; the SECURE EXTERNAL EXECUTION, and the VERIFY bit. Programming both bits denies any external access to the on-chip program memory.

The security bits, when programmed, prohibit the controller from reading or moving the internal code when executing out of external program memory, and also disable the verify mode.

It is possible to maintain the verify capability while securing the code. This is done by programming only one bit (SECURE EXTERNAL EXECUTION), and programming the SECURITY TABLE MEMORY. The data that appears on the port during the secured verification, is scrambled by the SECURITY TABLE contents.

In the 87C252 the SECURITY TABLE is a 32 x 8 array of EPROM. The user can electrically program any arbitrary code into it. The code will be used to mask the program memory during the verification. Thus what appears at port 0 during the verification is a scrambled code which cannot be deciphered without the key table.

In the 83C252 the Security Table and the Security Bits are mask programmable.

Programming the security bits also denies the programming of the EPROM and the Security Table Memory. Therefore they must be locked after the code and the security table are programmed.

Erasing the EPROM also erases the Security Table Memory and the security bits. When erased the EPROM and the table contain all 1s and the part has its full functionality.

## INTELLIGENT PROGRAMMING ALGORITHM

The 87C252 can be electrically programmed by using the Intelligent programming algorithm. This method is faster and more efficient than the conventional method. The Intelligent Programming algorithm reduces the programming time from 50 m sec per byte to a minimum of 4 m sec per byte.

## MEMORY ORGANIZATION

**PROGRAM MEMORY:** Up to 8K bytes of the program

memory can reside in the on-chip ROM. In addition the device can address up to 64K of program memory external to the chip.

**DATA MEMORY:** This microcontroller has a 256 x 8 on-chip RAM. In addition it can address up to 64K bytes of external data memory.

## PACKAGING INFORMATION

40 pin cerdip D8xC252  
44 pin lcc R8xC252

The security system of the 87C252 and the 83C252 is designed to give the user the maximum control in protecting the internal program memory of the part. It allows the user to apply the degree of security suitable for the application.

Two security bits are implemented: the SECURE EXTERNAL EXECUTION, and the VERIFY bit. Programming both bits denies any external access to the on-chip program memory.

The security bits, when programmed, prohibit the controller from reading or moving the internal code when executing out of external program memory, and also disable the verify mode.

It is possible to maintain the verify capability while securing the code. This is done by programming only one bit (SECURE EXTERNAL EXECUTION), and one bit (VERIFY). The SECURITY TABLE MEMORY. The data that appears on the port during the secured verification, is scrambled by the SECURITY TABLE contents.

In the 87C252 the SECURITY TABLE is a 32 x 8 array of EPROM. The user can electrically program any arbitrary code into it. The code will be used to mask the program memory during the verification. Thus what appears at port 0 during the verification is a scrambled code which cannot be deciphered without the key table.

In the 83C252 the Security Table and the Security Bits are mask programmable.

Programming the security bits also denies the programming of the EPROM and the Security Table Memory. Therefore they must be loaded after the code and the security table are programmed.

Erasing the EPROM also erases the Security Table Memory and the security bits. When erased the EPROM and the table contain all 1s and the bit has its full functionality.

## COMPARATOR/CAPTURE

When programmed as a comparator, the PCA module, at every cycle, compares the contents of the register with the present value of the capture register. If a match occurs, it reverses the logic level of its corresponding I/O pin, and generates an interrupt.

In the Capture mode the process is the reverse of the Comparator. Upon changing the logic level on the corresponding I/O pin of the module, the content of the PCA Timer is loaded into the selected Capture register. An interrupt may also be generated.

These two features allow the 80C151 to be used in accurate pulse width measurement in real time, with minimum software overhead.

## WATCHDOG TIMER

Module number 4 of the PCA, along with its Timer, can be programmed to play Watchdog Timer. If the timer value matches the Comparator Register value, an internal Reset signal is generated which puts the microcontroller in a hardware reset.

During normal operation the software, periodically reloads the timer or disables the internal reset signal. Therefore, in case of a malfunction due to noise or any unwanted situation, the processor recovers itself and provides a more reliable operation.

## TIMER/COUNTER

Timer 2 is a 16-bit timer/counter which is capable of up or down counting. It has a 16-bit register that can be used for auto-reloading or as a capture register. The capture command can come from external source as well as the clock input to the timer. With the help of Timer 2, two different baud rates can be generated for transmit and receive. Timer 2 can be utilized while the processor is in the Idle Mode.

## SERIAL PORT

The full duplex serial port of the 80C252 is the same as the serial port of the 80C51 with two new features: Framing Error Detection, and Automatic Address Recognition.

When the Serial Port is operating in either mode 2 or

# 87C51

## CHMOS SINGLE-CHIP 8-BIT MICROCONTROLLER WITH 4K BYTES OF EPROM PROGRAM MEMORY AND INTERNAL CODE SECURITY FEATURE

- High Performance CHMOS Process
- Power Control Modes
- 2-Level Program Security System
- 128 Bytes Data RAM
- intelligent Programming™ Algorithm
- 12.5 V Programming Voltage
- Boolean Processor
- 32 Programmable I/O Lines
- Two 16-Bit Timer/Counters
- 5 Interrupt Sources
- Programmable Serial Channel
- TTL Compatible Logic Levels
- 64K Program Memory Space
- 64K Data Memory Space
- LCC and DIP packagings available

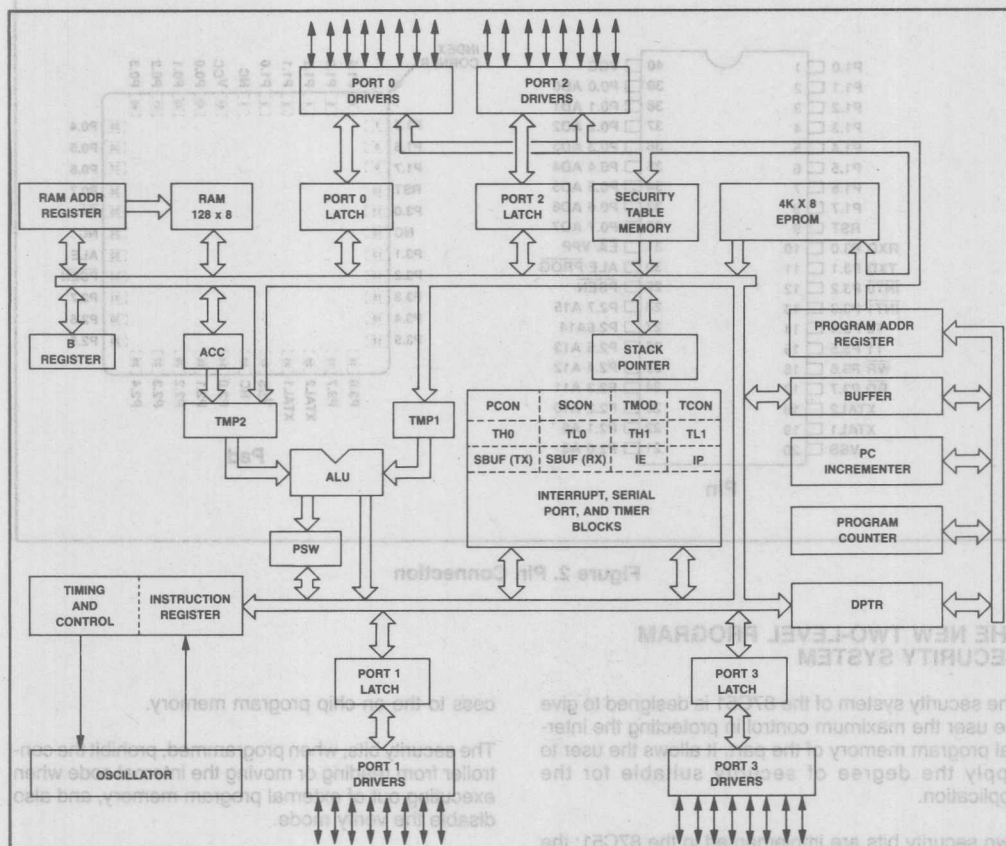


Figure 1. Block Diagram



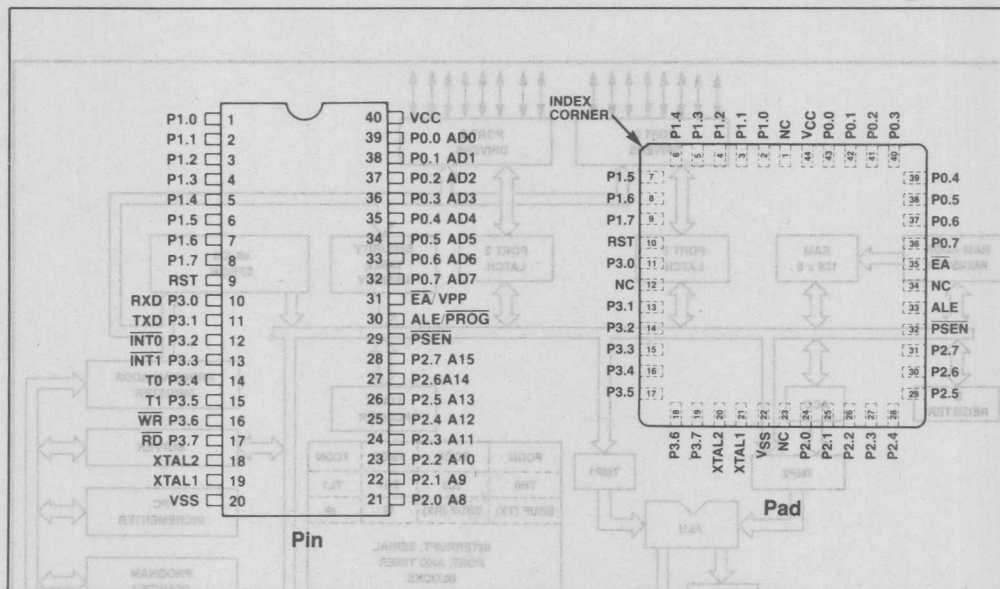
The Intel 87C51 is the EPROM version of the 80C51BH, and is fabricated on Intel's CHMOS II-E process. It contains 4K x 8 of on-chip Program memory that can be electrically programmed, and can be erased by exposure to ultraviolet light.

The 87C51 is the EPROM version of the 80C51BH and a member of the MCS<sup>®</sup>-51 family of microcontrollers. It is equipped with a 2-level program memory security system which protects the on-chip program against software piracy.

This EPROM device can be electrically programmed by means of the Intelligent Programming algorithm.

The extremely low power consumption, along with two reduced power modes (Idle and Power Down), make this part very suitable for low power applications.

The Idle mode freezes the CPU while allowing the RAM, Timer/Counters, serial port, and interrupt system to continue functioning. The Power Down mode saves the RAM contents but freezes the oscillator, causing all other chip functions to be inoperative.



### Figure 2. Pin Connection

## THE NEW TWO-LEVEL PROGRAM SECURITY SYSTEM

The security system of the 87C51 is designed to give the user the maximum control in protecting the internal program memory of the part. It allows the user to apply the degree of security suitable for the application.

Two security bits are implemented in the 87C51; the SECURE EXTERNAL EXECUTION, and the VERIFY bit. Programming both bits denies any external ac-

cess to the on-chip program memory.

The security bits, when programmed, prohibit the controller from reading or moving the internal code when executing out of external program memory, and also disable the verify mode.

It is possible to maintain the verify capability while securing the code. This is done by programming only



one bit (SECURE EXTERNAL EXECUTION), and programming the SECURITY TABLE. The data that appears on the port during the secured verification is scrambled by the SECURITY TABLE contents.

The SECURITY TABLE is a 32 x 8 array of EPROM. The user can electrically program any arbitrary code into it. The code will be used to mask the program memory during the verification. Thus what appears at port 0 during the verification is a scrambled code which cannot be deciphered without the key table.

Programming the security bits also denies the programming of the EPROM and the Security Table Memory. Therefore, they must be locked after the code and the security table are programmed.

Erasing the EPROM also erases the Security Table Memory and the security bits. When erased, the EPROM and the table contain all 1s and the part has its full functionality.

#### **INTELLIGENT PROGRAMMING ALGORITHM**

The 87C51 can be electrically programmed by using

the Intelligent programming algorithm. This method is faster and more efficient than the conventional method. The Intelligent Programming algorithm reduces the programming time from 50 msec per byte to a minimum of 4 msec per byte.

#### **MEMORY ORGANIZATION**

**PROGRAM MEMORY:** Up to 4K bytes of the program memory can reside in the on-chip EPROM. In addition the device can address up to 64K of program memory external to the chip.

**DATA MEMORY:** This microcontroller has a 128 x 8 on-chip RAM. In addition it can address up to 64K bytes of external data memory.

#### **PACKAGING INFORMATION**

|               |        |
|---------------|--------|
| 40 pin cerdip | D87C51 |
| 44 pin lcc    | R87C51 |



# Intel MCS-51 Boolean Processing Capabilities

|   |       |
|---|-------|
| 1. INTRODUCTION . . . . .                         | 10-32 |
| 2. BOOLEAN PROCESSOR OPERATION . . . . .          | 10-32 |
| Processing Elements . . . . .                     | 10-32 |
| Direct Bit Addressing . . . . .                   | 10-34 |
| Instruction Set . . . . .                         | 10-39 |
| Simple Instruction Combinations . . . . .         | 10-40 |
| 3. BOOLEAN PROCESSOR APPLICATIONS . . . . .       | 10-41 |
| Design Example #1 — Bit Permutation . . . . .     | 10-42 |
| Design Example #2 — Software Serial I/O . . . . . | 10-45 |
| Design Example #3 — . . . . .                     | 10-45 |
| Combinational Logic Equations . . . . .           | 10-46 |
| Design Example #4 — . . . . .                     | 10-46 |
| Automotive Dashboard Functions . . . . .          | 10-49 |
| Design Example #5 — . . . . .                     | 10-49 |
| Complex Control Functions . . . . .               | 10-54 |
| Additional Functions and Uses . . . . .           | 10-59 |
| 4. SUMMARY . . . . .                              | 10-60 |
| APPENDIX A . . . . .                              | 10-61 |

# 01

## Using the Intel MCS®-51 Boolean Processing Capabilities

### Contents

|   |       |
|---|-------|
| 1. INTRODUCTION . . . . .                                       | 10-32 |
| 2. BOOLEAN PROCESSOR OPERATION . . . . .                        | 10-32 |
| Processing Elements. . . . .                                    | 10-33 |
| Direct Bit Addressing. . . . .                                  | 10-34 |
| Instruction Set. . . . .  | 10-39 |
| Simple Instruction Combinations . . . . .                       | 10-40 |
| 3. BOOLEAN PROCESSOR APPLICATIONS . . . . .                     | 10-41 |
| Design Example #1 — Bit Permutation. . . . .                    | 10-42 |
| Design Example #2 — Software Serial I/O. . . . .                | 10-45 |
| Design Example #3 —<br>Combinatorial Logic Equations. . . . .   | 10-46 |
| Design Example #4 —<br>Automotive Dashboard Functions . . . . . | 10-49 |
| Design Example #5 —<br>Complex Control Functions. . . . .       | 10-54 |
| Additional Functions and Uses . . . . .                         | 10-59 |
| 4. SUMMARY . . . . .  | 10-60 |
| APPENDIX A . . . . .  | 10-61 |

## 1. INTRODUCTION

The Intel microcontroller family now has three new members—the Intel® 8031, 8051, and 8751 single-chip microcomputers. These devices, shown in Figure 1, will allow whole new classes of products to benefit from recent advances in Integrated Electronics. Thanks to Intel's new HMOS® technology, they provide larger program and data memory spaces, more flexible I/O and peripheral capabilities, greater speed, and lower system cost than any previous-generation single-chip microcomputer.

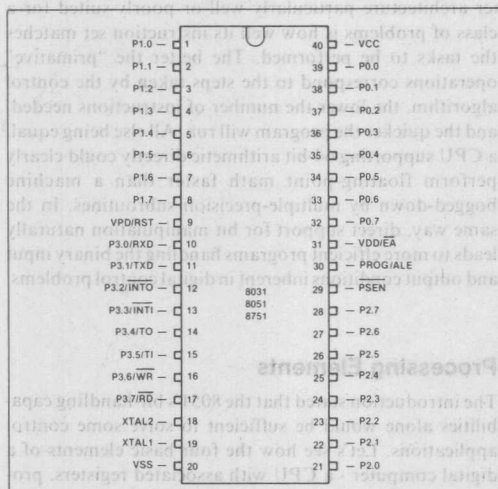


Figure 1. 8051 Family Pinout Diagram.

Table 1 summarizes the quantitative differences between the members of the MCS-48™ and 8051 families. The 8751 contains 4K bytes of EPROM program memory fabricated on-chip, while the 8051 replaces the EPROM with 4K bytes of lower-cost mask-programmed ROM. The 8031 has no program memory on-chip; instead, it accesses up to 64K bytes of program memory from external memory. Otherwise, the three new family members are identical. Throughout this Note, the term "8051" will represent all members of the 8051 Family, unless specifically stated otherwise.

Table 1. Features of Intel's Single-chip Microcomputers.

| EPROM<br>Program<br>Memory | ROM<br>Program<br>Memory | External<br>Program<br>Memory | Program<br>Memory<br>(Int/Max) | Data<br>Memory<br>(Bytes) | Instr.<br>Cycle<br>Time | Input/<br>Output<br>Pins | Interrupt<br>Sources | Reg.<br>Banks |
|----------------------------|--------------------------|-------------------------------|--------------------------------|---------------------------|-------------------------|--------------------------|----------------------|---------------|
|                            | 8021                     |                               | 1K 1K                          | 64                        | 10 µSec                 | 21                       | 0                    | 1             |
|                            | 8022                     |                               | 2K 2K                          | 64                        | 10 µSec                 | 28                       | 2                    | 1             |
| 8748                       | 8048                     | 8035                          | 1K 4K                          | 64                        | 2.5 µSec                | 27                       | 2                    | 2             |
|                            | 8049                     | 8039                          | 2K 4K                          | 128                       | 1.36 µSec               | 27                       | 2                    | 2             |
| 8751                       | 8051                     | 8031                          | 4K 64K                         | 128                       | 1.0 µSec                | 32                       | 5                    | 4             |

The CPU in each microcomputer is one of the industry's fastest and most efficient for numerical calculations on byte operands. But controllers often deal with bits, not bytes: in the real world, switch contacts can only be open or closed, indicators should be either lit or dark, motors are either turned on or off, and so forth. For such control situations the most significant aspect of the MCS-51™ architecture is its complete hardware support for one-bit, or *Boolean* variables (named in honor of Mathematician George Boole) as a separate data type.

The 8051 incorporates a number of special features which support the direct manipulation and testing of individual bits and allow the use of single-bit variables in performing logical operations. Taken together, these features are referred to as the MCS-51™ *Boolean Processor*. While the bit-processing capabilities alone would be adequate to solve many control applications, their true power comes when they are used in conjunction with the microcomputer's byte-processing and numerical capabilities.

Many concepts embodied by the Boolean Processor will certainly be new even to experienced microcomputer system designers. The purpose of this Application Note is to explain these concepts and show how they are used. It is assumed the reader has read Application Note AP-69, **An Introduction to the Intel® MCS-51™ Single-Chip Microcomputer Family**, publication number 121518, or has been exposed to Intel's single-chip microcomputer product lines.

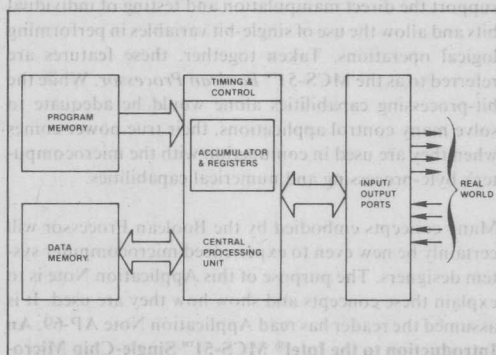
For detailed information on these parts refer to the **Intel MCS-51™ Family User's Manual**, publication number 121517. The instruction set, assembly language, and use of the 8051 assembler (ASM51) are further described in the **MCS-51™ Macro Assembler User's Guide**, publication number 9800937.

## 2. BOOLEAN PROCESSOR OPERATION

The Boolean Processing capabilities of the 8051 are based on concepts which have been around for some time. Digital computer systems of widely varying designs all have four functional elements in common (Figure 2):



- a central processor (CPU) with the control, timing, and logic circuits needed to execute stored instructions;
- a memory to store the sequence of instructions making up a program or algorithm;
- data memory to store variables used by the program; and
- some means of communicating with the outside world.



**Figure 2. Block Diagram for Abstract Digital Computer.**

The CPU usually includes one or more accumulators or special registers for computing or storing values during program execution. The instruction set of such a processor generally includes, at a minimum, operation classes to perform arithmetic or logical functions on program variables, move variables from one place to another, cause program execution to jump or conditionally branch based on register or variable states, and instructions to call and return from subroutines. The program and data memory functions sometimes share a single memory space, but this is not always the case. When the address spaces are separated, program and data memory need not even have the same basic word width.

A digital computer's flexibility comes in part from combining simple fast operations to produce more complex (albeit slower) ones, which in turn link together eventually solving the problem at hand. A four-bit CPU executing multiple precision subroutines can, for example, perform 64-bit addition and subtraction. The subroutines could in turn be building blocks for floating-point multiplication and division routines. Eventually, the four-bit CPU can simulate a far more complex "virtual" machine.

In fact, *any* digital computer with the above four functional elements can (given time) complete *any* algorithm (though the proverbial room full of chimpanzees at word

processors might first re-create Shakespeare's classics and this Application Note)! This fact offers little consolation to product designers who want programs to run as quickly as possible. By definition, a real-time control algorithm *must* proceed quickly enough to meet the preordained speed constraints of other equipment.

One of the factors determining how long it will take a microcomputer to complete a given chore is the number of instructions it must execute. What makes a given computer architecture particularly well- or poorly-suited for a class of problems is how well its instruction set matches the tasks to be performed. The better the "primitive" operations correspond to the steps taken by the control algorithm, the lower the number of instructions needed, and the quicker the program will run. All else being equal, a CPU supporting 64-bit arithmetic directly could clearly perform floating-point math faster than a machine bogged-down by multiple-precision subroutines. In the same way, direct support for bit manipulation naturally leads to more efficient programs handling the binary input and output conditions inherent in digital control problems.

## Processing Elements

The introduction stated that the 8051's bit-handling capabilities alone would be sufficient to solve some control applications. Let's see how the four basic elements of a digital computer - a CPU with associated registers, program memory, addressable data RAM, and I/O capability - relate to Boolean variables:

**CPU.** The 8051 CPU incorporates special logic devoted to executing several bit-wide operations. All told, there are 17 such instructions, all listed in Table 2. Not shown are 94 other (mostly byte-oriented) 8051 instructions.

**Program Memory.** Bit-processing instructions are fetched from the same program memory as other arithmetic and logical operations. In addition to the instructions of Table 2, several sophisticated program control features like multiple addressing modes, subroutine nesting, and a two-level interrupt structure are useful in structuring Boolean Processor-based programs.

Boolean instructions are one, two, or three bytes long, depending on what function they perform. Those involving only the carry flag have either a single-byte opcode or an opcode followed by a conditional-branch destination byte (Figure 3.a). The more general instructions add a "direct address" byte after the opcode to specify the bit affected, yielding two or three byte encodings (Figure 3.b). Though this format allows potentially 256 directly addressable bit locations, not all of them are implemented in the 8051 family.

**Table 2. MCS-51™ Boolean Processing Instruction Subset.**

| Mnemonic    | Description                                | Byte | Cyc |
|-------------|--|------|-----|
| SETB C      | Set Carry flag                             | 1    | 1   |
| SETB bit    | Set direct Bit                             | 2    | 1   |
| CLR C       | Clear Carry flag                           | 1    | 1   |
| CLR bit     | Clear direct bit                           | 2    | 1   |
| CPL C       | Complement Carry flag                      | 1    | 1   |
| CPL bit     | Complement direct bit                      | 2    | 1   |
| MOV C,bit   | Move direct bit to Carry flag              | 2    | 1   |
| MOV bit,C   | Move Carry flag to direct bit              | 2    | 2   |
| ANL C,bit   | AND direct-bit to Carry flag               | 2    | 2   |
| ANL C, bit  | AND complement of direct bit to Carry flag | 2    | 2   |
| ORL C,bit   | OR direct bit to Carry flag                | 2    | 2   |
| ORL C, bit  | OR complement of direct bit to Carry flag  | 2    | 2   |
| JC rel      | Jump if Carry flag is set                  | 2    | 2   |
| JNC rel     | Jump if No Carry flag                      | 2    | 2   |
| JB bit,rel  | Jump if direct Bit set                     | 3    | 2   |
| JNB bit,rel | Jump if direct Bit Not set                 | 3    | 2   |
| JBC bit,rel | Jump if direct Bit is set & Clear bit      | 3    | 2   |

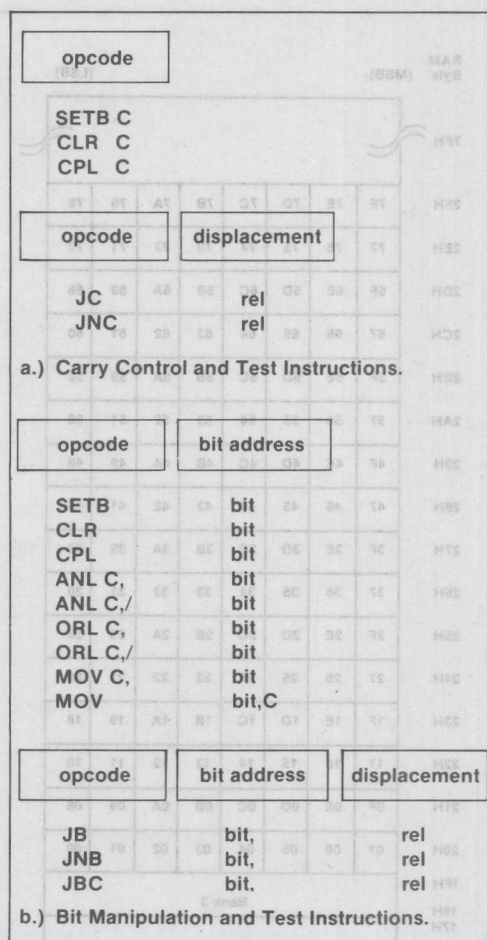
**Address mode abbreviations:**

|     |   |
|-----|---|
| C   | Carry flag.   |
| bit | 128 software flags, any I/O pin, control or status bit  |
| rel | All conditional jumps include an 8-bit offset byte. Range is +127 -128 bytes relative to first byte of the following instruction. |

All mnemonics copyrighted © Intel Corporation 1980

**Data Memory.** The instructions in Figure 3.b can operate directly upon 144 general-purpose bits forming the Boolean processor "RAM." These bits can be used as software flags or to store program variables. Two operand instructions use the CPU's carry flag ("C") as a special one-bit register; in a sense, the carry is a "Boolean accumulator" for logical operations and data transfers.

**Input/Output.** All 32 I/O pins can be addressed as individual inputs, outputs, or both, in any combination. Any pin can be a control strobe output, status (Test) input, or serial I/O link implemented via software. An additional 33 individually addressable bits reconfigure, control, and monitor the status of the CPU and all on-chip peripheral functions (timer counters, serial port modes, interrupt logic, and so forth).



**Figure 3. Bit Addressing Instruction Formats.**

### Direct Bit Addressing

The most significant bit of the direct address byte selects one of two groups of bits. Values between 0 and 127 (00H and 7FH) define bits in a block of 32 bytes of on-chip RAM, between RAM addresses 20H and 2FH (Figure 4.a). They are numbered consecutively from the lowest-order byte's lowest-order bit through the highest-order byte's highest-order bit.

Bit addresses between 128 and 255 (80H and 0FFH) correspond to bits in a number of special registers, mostly used for I/O or peripheral control. These positions are numbered with a different scheme than RAM: the five high-order address bits match those of the register's own address, while the three low-order bits identify the bit position within that register (Figure 4.b).

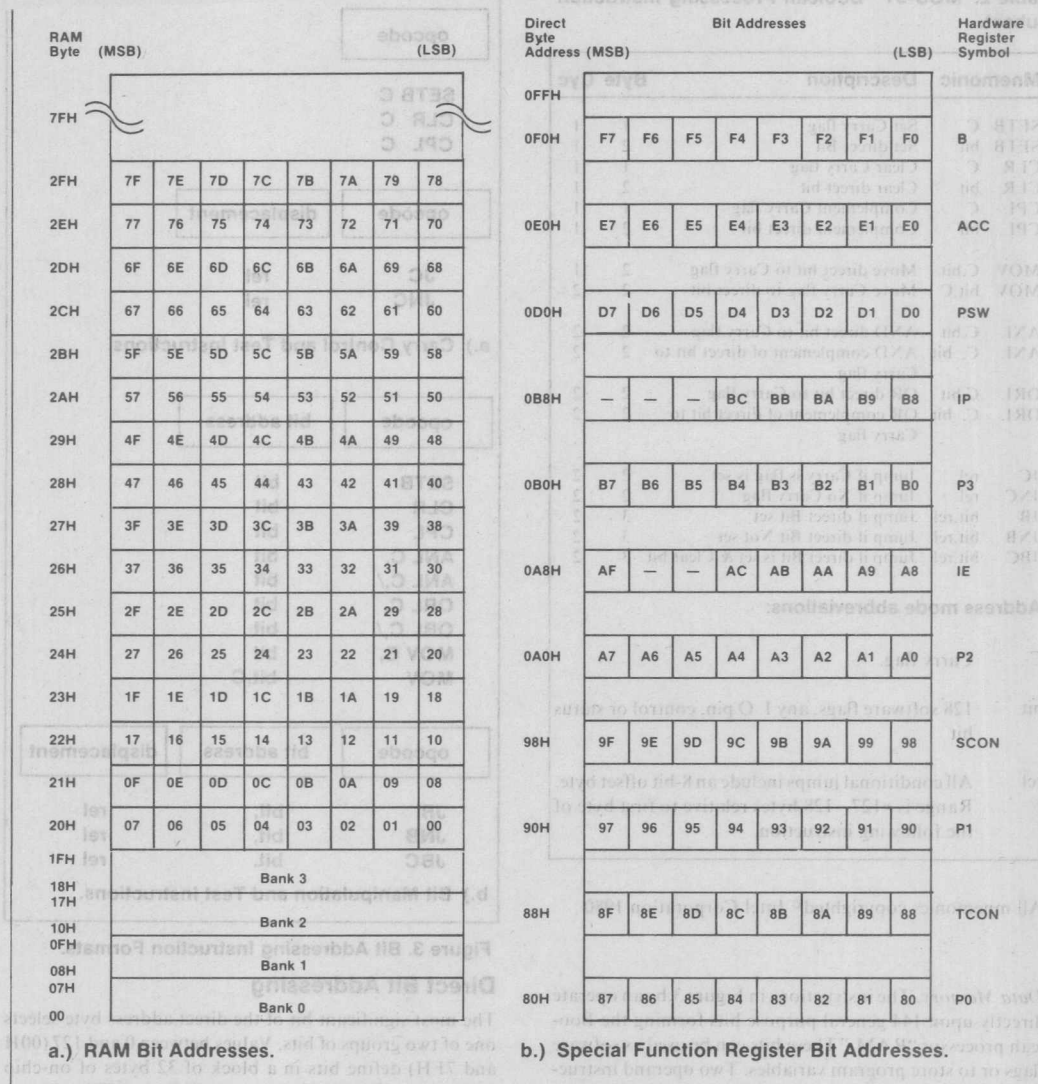


Figure 4. Bit Address Maps.

Notice the column labeled "Symbol" in Figure 5. Bits with special meanings in the PSW and other registers have corresponding symbolic names. General-purpose (as opposed to carry-specific) instructions may access the carry like any other bit by using the mnemonic CY in place of C. P0, P1, P2, and P3 are the 8051's four I/O ports; secondary functions assigned to each of the eight pins of P3 are shown in Figure 6.

Figure 7 shows the last four bit addressable registers. TCON (Timer Control) and SCON (Serial port Control) control and monitor the corresponding peripherals, while IE (Interrupt Enable) and IP (Interrupt Priority) enable and prioritize the five hardware interrupt sources. Like the reserved hardware register addresses, the five bits not implemented in IE and IP should not be accessed; they can not be used as software flags.

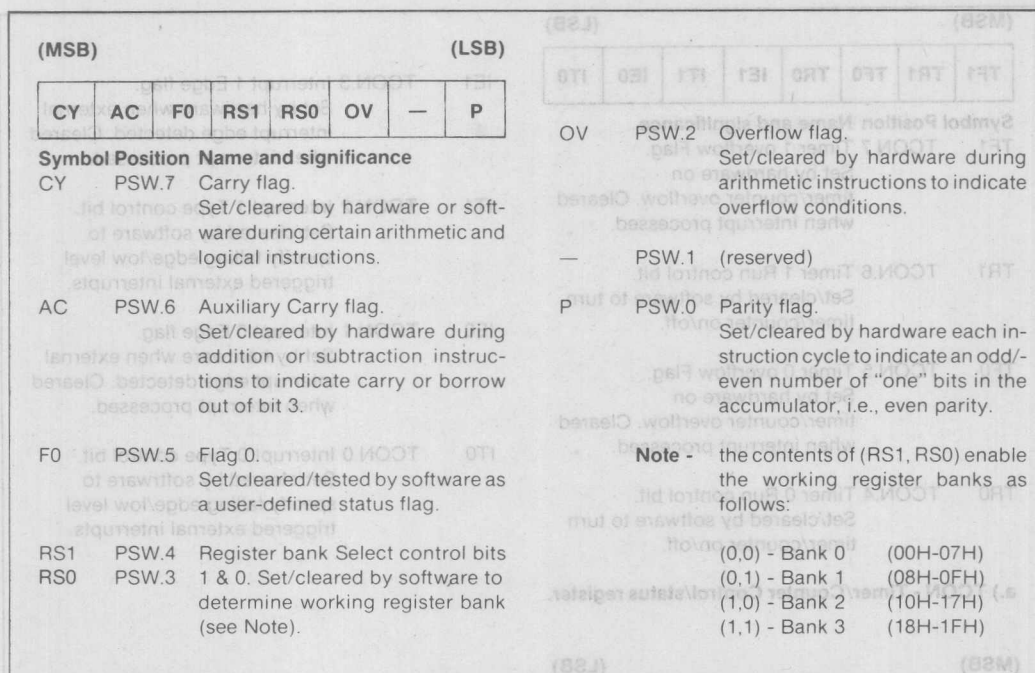


Figure 5. PSW - Program Status Word organization.

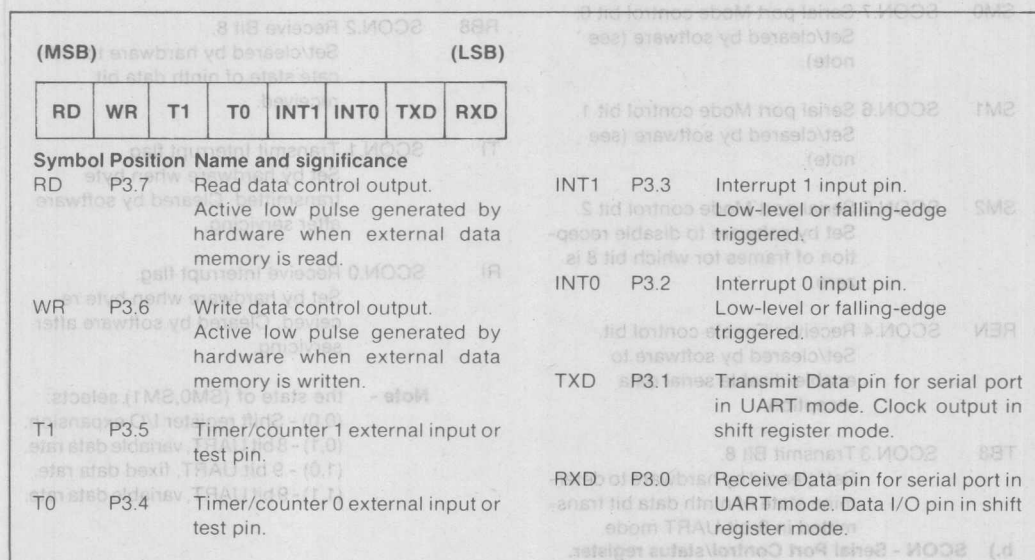


Figure 6. P3 - Alternate I/O Functions of Port 3.



| (MSB)  |        |   |     |     |     |     |     | (LSB) |        |   |     |  |  |  |  |
|--|--------|---|-----|-----|-----|-----|-----|-------|--------|---|-----|--|--|--|--|
| TF1  | TR1    | TF0   | TR0 | IE1 | IT1 | IE0 | IT0 | IE1   | IT1    | IE0   | IT0 |  |  |  |  |
| <b>Symbol Position Name and significance</b>   |        |   |     |     |     |     |     |       |        |   |     |  |  |  |  |
| TF1  | TCON.7 | Timer 1 overflow Flag.<br>Set by hardware on timer/counter overflow. Cleared when interrupt processed.          |     |     |     |     |     |       |        |   |     |  |  |  |  |
| TR1  | TCON.6 | Timer 1 Run control bit.<br>Set/cleared by software to turn timer/counter on/off.                               |     |     |     |     |     |       |        |   |     |  |  |  |  |
| TF0  | TCON.5 | Timer 0 overflow Flag.<br>Set by hardware on timer/counter overflow. Cleared when interrupt processed.          |     |     |     |     |     |       |        |   |     |  |  |  |  |
| TR0  | TCON.4 | Timer 0 Run control bit.<br>Set/cleared by software to turn timer/counter on/off.                               |     |     |     |     |     |       |        |   |     |  |  |  |  |
| <b>a.) TCON - Timer/Counter Control/status register.</b>   |        |   |     |     |     |     |     |       |        |   |     |  |  |  |  |
| (MSB)  |        |   |     |     |     |     |     | (LSB) |        |   |     |  |  |  |  |
| SM0  | SM1    | SM2   | REN | TB8 | RB8 | TI  | RI  | RB8   | SCON.2 | TI  | RI  |  |  |  |  |
| <b>Symbol Position Name and significance</b>   |        |   |     |     |     |     |     |       |        |   |     |  |  |  |  |
| SM0  | SCON.7 | Serial port Mode control bit 0.<br>Set/cleared by software (see note).  |     |     |     |     |     |       |        |   |     |  |  |  |  |
| SM1  | SCON.6 | Serial port Mode control bit 1.<br>Set/cleared by software (see note).  |     |     |     |     |     |       |        |   |     |  |  |  |  |
| SM2  | SCON.5 | Serial port Mode control bit 2.<br>Set by software to disable reception of frames for which bit 8 is zero.      |     |     |     |     |     |       |        |   |     |  |  |  |  |
| REN  | SCON.4 | Receiver Enable control bit.<br>Set/cleared by software to enable/disable serial data reception.                |     |     |     |     |     |       |        |   |     |  |  |  |  |
| TB8  | SCON.3 | Transmit Bit 8.<br>Set/cleared by hardware to determine state of ninth data bit transmitted in 9-bit UART mode. |     |     |     |     |     |       |        |   |     |  |  |  |  |
| <b>b.) SCON - Serial Port Control/status register.</b>   |        |   |     |     |     |     |     |       |        |   |     |  |  |  |  |
|  |        |   |     |     |     |     |     | RB8   | SCON.2 | Receive Bit 8.<br>Set/cleared by hardware to indicate state of ninth data bit received.                 |     |  |  |  |  |
|  |        |   |     |     |     |     |     | TI    | SCON.1 | Transmit Interrupt flag.<br>Set by hardware when byte transmitted. Cleared by software after servicing. |     |  |  |  |  |
|  |        |   |     |     |     |     |     | RI    | SCON.0 | Receive Interrupt flag.<br>Set by hardware when byte received. Cleared by software after servicing.     |     |  |  |  |  |
| <b>Note -</b> the state of (SM0,SM1) selects:<br>(0,0) - Shift register I/O expansion.<br>(0,1) - 8 bit UART, variable data rate.<br>(1,0) - 9 bit UART, fixed data rate.<br>(1,1) - 9 bit UART, variable data rate. |        |   |     |     |     |     |     |       |        |   |     |  |  |  |  |

Figure 7. Peripheral Configuration Registers.

Figure 7. Peripheral Configuration Registers.



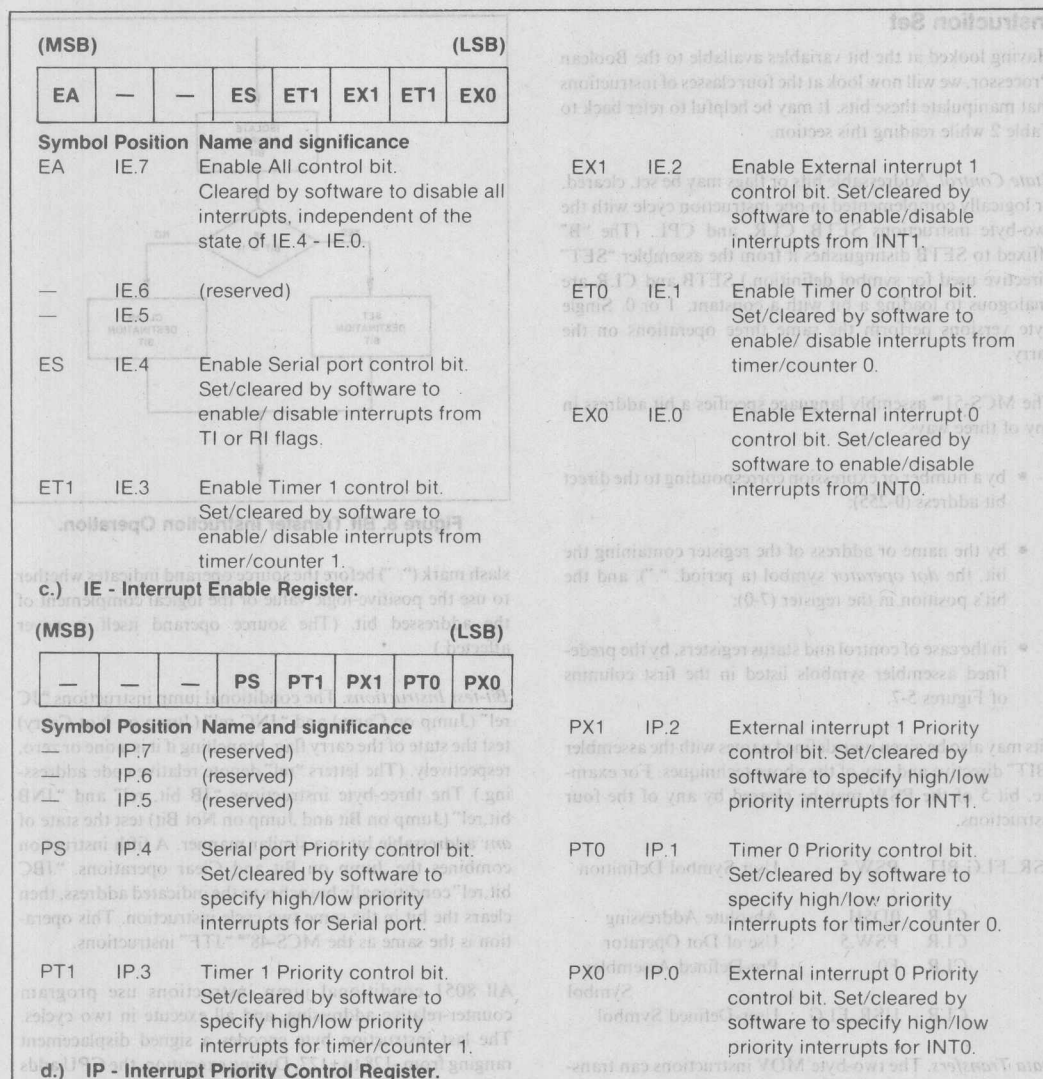


Figure 7. (continued)

**Addressable Register Set.** There are 20 special function registers in the 8051, but the advantages of bit addressing only relate to the 11 described below. Five potentially bit-addressable register addresses (0C0H, 0C8H, 0D8H, 0E8H, & 0F8H) are being reserved for possible future expansion in microcomputers based on the MCS-51™ architecture. Reading or writing non-existent registers in the 8051 series is pointless, and may cause unpredictable results. Byte-wide logical operations can be used to manipulate bits in all non-bit addressable registers and RAM.

The accumulator and B registers (A and B) are normally involved in byte-wide arithmetic, but their individual bits can also be used as 16 general software flags. Added with the 128 flags in RAM, this gives 144 general purpose variables for bit-intensive programs. The program status word (PSW) in Figure 5 is a collection of flags and machine status bits including the carry flag itself. Byte operations acting on the PSW can therefore affect the carry.

## Instruction Set

Having looked at the bit variables available to the Boolean Processor, we will now look at the four classes of instructions that manipulate these bits. It may be helpful to refer back to Table 2 while reading this section.

**State Control.** Addressable bits or flags may be set, cleared, or logically complemented in one instruction cycle with the two-byte instructions SETB, CLR, and CPL. (The "B" affixed to SETB distinguishes it from the assembler "SET" directive used for symbol definition.) SETB and CLR are analogous to loading a bit with a constant: 1 or 0. Single byte versions perform the same three operations on the carry.

The MCS-51™ assembly language specifies a bit address in any of three ways:

- by a number or expression corresponding to the direct bit address (0-255);
- by the name or address of the register containing the bit, the *dot operator* symbol (a period: "."), and the bit's position in the register (7-0);
- in the case of control and status registers, by the predefined assembler symbols listed in the first columns of Figures 5-7.

Bits may also be given user-defined names with the assembler "BIT" directive and any of the above techniques. For example, bit 5 of the PSW may be cleared by any of the four instructions.

USR\_FLG BIT PSW.5 : User Symbol Definition

CLR 0D5H : Absolute Addressing

CLR PSW.5 : Use of Dot Operator

CLR F0 : Pre-Defined Assembler

Symbol

CLR USR\_FLG : User-Defined Symbol

**Data Transfers.** The two-byte MOV instructions can transport any addressable bit to the carry in one cycle, or copy the carry to the bit in two cycles. A bit can be moved between two arbitrary locations via the carry by combining the two instructions. (If necessary, push and pop the PSW to preserve the previous contents of the carry.) These instructions can replace the multi-instruction sequence of Figure 8, a program structure appearing in controller applications whenever flags or outputs are conditionally switched on or off.

**Logical Operations.** Four instructions perform the logical-AND and logical-OR operations between the carry and another bit, and leave the results in the carry. The instruction mnemonics are ANL and ORL; the absence or presence of a

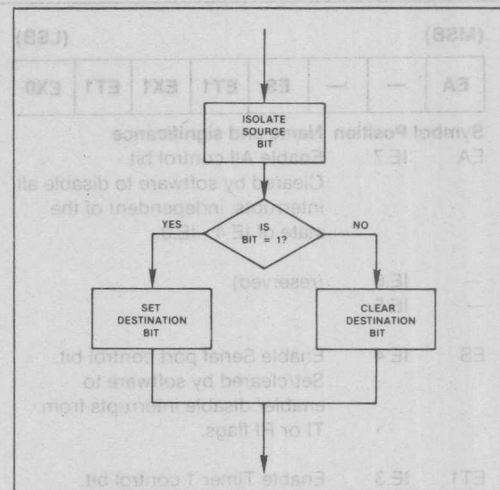


Figure 8. Bit Transfer Instruction Operation.

slash mark ("/) before the source operand indicates whether to use the positive-logic value or the logical complement of the addressed bit. (The source operand itself is never affected.)

**Bit-test Instructions.** The conditional jump instructions "JC rel" (Jump on Carry) and "JNC rel" (Jump on Not Carry) test the state of the carry flag, branching if it is a one or zero, respectively. (The letters "rel" denote relative code addressing.) The three-byte instructions "JB bit, rel" and "JNB bit, rel" (Jump on Bit and Jump on Not Bit) test the state of any addressable bit in a similar manner. A fifth instruction combines the Jump on Bit and Clear operations. "JBC bit, rel" conditionally branches to the indicated address, then clears the bit in the same two cycle instruction. This operation is the same as the MCS-48™ "JTF" instructions.

All 8051 conditional jump instructions use program counter-relative addressing, and all execute in two cycles. The last instruction byte encodes a signed displacement ranging from -128 to +127. During execution, the CPU adds this value to the incremented program counter to produce the jump destination. Put another way, a conditional jump to the immediately following instruction would encode 00H in the offset byte.

A section of program or subroutine written using only relative jumps to nearby addresses will have the same machine code independent of the code's location. An assembled routine may be repositioned anywhere in memory, even crossing memory page boundaries, without having to modify the program or recompute destination addresses. To facilitate this flexibility, there is an unconditional "Short Jump" (SJMP) which uses relative addressing as well. Since a pro-

programmer would have quite a chore trying to compute relative offset values from one instruction to another. ASM51 automatically computes the displacement needed given only the destination address or label. An error message will alert the programmer if the destination is "out of range."

(The so-called "Bit Test" instructions implemented on many other microprocessors simply perform the logical-AND operation between a byte variable and a constant mask, and set or clear a zero flag depending on the result. This is essentially equivalent to the 8051 "MOV C,bit" instruction. A second instruction is then needed to conditionally branch based on the state of the zero flag. This does *not* constitute abstract bit-addressing in the MCS-51™ sense. A flag exists only as a field within a register; to reference a bit the programmer must know and specify both the encompassing register and the bit's position therein. This constraint severely limits the flexibility of symbolic bit addressing and reduces the machine's code-efficiency and speed.)

**Interaction with Other Instructions.** The carry flag is also affected by the instructions listed in Table 3. It can be rotated through the accumulator, and altered as a side effect of arithmetic instructions. Refer to the User's Manual for details on how these instructions operate.

### Simple Instruction Combinations

By combining general purpose bit operations with certain addressable bits, one can "custom build" several hundred useful instructions. All eight bits of the PSW can be tested directly with conditional jump instructions to monitor (among other things) parity and overflow status. Programmers can take advantage of 128 software flags to keep track of operating modes, resource usage, and so forth.

The Boolean instructions are also the most efficient way to control or reconfigure peripheral and I/O registers. All 32 I/O lines become "test pins," for example, tested by conditional jump instructions. Any output pin can be toggled (complemented) in a single instruction cycle. Setting or clearing the Timer Run flags (TR0 and TR1) turn the timer-counters on or off; polling the same flags elsewhere lets the program determine if a timer is running. The respective overflow flags (TF0 and TF1) can be tested to determine when the desired period or count has elapsed, then cleared in preparation for the next repetition. (For the record, these bits are all part of the TCON register. Figure 7.a. Thanks to symbolic bit addressing, the programmer only needs to remember the mnemonic associated with each function. In other words, don't bother memorizing control word layouts.)

In the MCS-48® family, instructions corresponding to some of the above functions require specific opcodes. Ten different opcodes serve to clear/complement the software flags F0 and F1, enable/disable each interrupt, and start/stop the timer. In the 8051 instruction set, just three opcodes (SETB,

**Table 3. Other Instructions Affecting the Carry Flag.**

| Mnemonic           | Description                                     | Byte | Cyc |
|--------------------|---|------|-----|
| ADD A,Rn           | Add register to Accumulator                     | 1    | 1   |
| ADD A,direct       | Add direct byte to Accumulator                  | 2    | 1   |
| ADD A,@Ri          | Add indirect RAM to Accumulator                 | 1    | 1   |
| ADD A,#data        | Add immediate data to Accumulator               | 2    | 1   |
| ADDC A,Rn          | Add register to Accumulator with Carry flag     | 1    | 1   |
| ADDC A,direct      | Add direct byte to Accumulator with Carry flag  | 2    | 1   |
| ADDC A,@Ri         | Add indirect RAM to Accumulator with Carry flag | 1    | 1   |
| ADDC A,#data       | Add immediate data to Acc with Carry flag       | 2    | 1   |
| SUBB A,Rn          | Subtract register from Accumulator with borrow  | 1    | 1   |
| SUBB A,direct      | Subtract direct byte from Acc with borrow       | 2    | 1   |
| SUBB A,@Ri         | Subtract indirect RAM from Acc with borrow      | 1    | 1   |
| SUBB A,#data       | Subtract immediate data from Acc with borrow    | 2    | 1   |
| MUL AB             | Multiply A & B                                  | 1    | 4   |
| DIV AB             | Divide A by B                                   | 1    | 4   |
| DA A               | Decimal Adjust Accumulator                      | 1    | 1   |
| RLC A              | Rotate Accumulator Left through the Carry flag  | 1    | 1   |
| RRC A              | Rotate Accumulator Right through Carry flag     | 1    | 1   |
| CJNE A,direct,rel  | Compare direct byte to Acc & Jump if Not Equal  | 3    | 2   |
| CJNE A,#data,rel   | Compare immediate to Acc & Jump if Not Equal    | 3    | 2   |
| CJNE Rn,#data,rel  | Compare immed to register & Jump if Not Equal   | 3    | 2   |
| CJNE @Ri,#data,rel | Compare immed to indirect & Jump if Not Equal   | 3    | 2   |

All mnemonics copyrighted © Intel Corporation 1980

CLR, CPL) with a direct bit address appended perform the same functions. Two test instructions (JB and JNB) can be combined with bit addresses to test the software flags, the 8048 I/O pins T0, T1, and INT, and the eight accumulator bits, replacing 15 more different instructions.

Table 4.a shows how 8051 programs implement software flag and machine control functions associated with special

using awkward sequences of other basic operations. As mentioned earlier, any CPU can solve any problem given enough time.

*Quantitatively*, the differences between a solution allowed by the 8051 and those required by previous architectures are numerous. What the 8051 Family buys you is a faster, cleaner, lower-cost solution to microcontroller applications.

The opcode space freed by condensing many specific 8048

**Table 4.a. Contrasting 8048 and 8051 Bit Control and Testing Instructions.**

| 8048<br>Instruction            | Bytes | Cycles | uSec | 8x51<br>Instruction | Bytes | Cycles & uSec |
|--------------------------------|-------|--------|------|---------------------|-------|---------------|
| Flag Control                   |       |        |      |                     |       |               |
| CLR C                          | 1     | 1      | 2.5  | CLR C               | 1     | 1             |
| CPL F0                         | 1     | 1      | 2.5  | CPL F0              | 2     | 1             |
| Flag Testing                   |       |        |      |                     |       |               |
| JNC offset                     | 2     | 2      | 5.0  | JNC rel             | 2     | 2             |
| JF0 offset                     | 2     | 2      | 5.0  | JB F0,rel           | 3     | 2             |
| JB7 offset                     | 2     | 2      | 5.0  | JB ACC.7,rel        | 3     | 2             |
| Peripheral Polling             |       |        |      |                     |       |               |
| JT0 offset                     | 2     | 2      | 5.0  | JB T0,rel           | 3     | 2             |
| JNI offset                     | 2     | 2      | 5.0  | JNB INT0,rel        | 3     | 2             |
| JTF offset                     | 2     | 2      | 5.0  | JBC TF0,rel         | 3     | 2             |
| Machine and Peripheral Control |       |        |      |                     |       |               |
| STRT T                         | 1     | 1      | 2.5  | SETB TR0            | 2     | 1             |
| EN I                           | 1     | 1      | 2.5  | SETB EX0            | 2     | 1             |
| DIS TCNTI                      | 1     | 1      | 2.5  | CLR ET0             | 2     | 1             |

**Table 4.b. Replacing 8048 instruction sequences with single 8x51 instructions.**

| 8048<br>Instructions | Bytes | Cycles | uSec | 8051<br>Instructions | Bytes | Cycles & uSec |
|----------------------|-------|--------|------|----------------------|-------|---------------|
| Flag Control         |       |        |      |                      |       |               |
| Set carry:           |       |        |      |                      |       |               |
| CLR C                | 1     | 1      | 2.5  | SETB C               | 1     | 1             |
| CPL C                | 1     | 1      | 2.5  |                      |       |               |
| Set Software Flag:   |       |        |      |                      |       |               |
| CLR F0               | 1     | 1      | 2.5  | SETB F0              | 2     | 1             |
| CPL F0               | 1     | 1      | 2.5  |                      |       |               |

opcodes in the 8048. In every case the MCS-51™ solution requires the same number of machine cycles, and executes 2.5 times faster.

### 3. BOOLEAN PROCESSOR APPLICATIONS

So what? Then what does all this buy you?

*Qualitatively*, nothing. All the same capabilities *could* be (and often have been) implemented on other machines

instructions into a few general operations has been used to add new functionality to the MCS-51™ architecture - both for byte and bit operations. 144 software flags replace the 8048's two. These flags (and the carry) may be directly set, not just cleared and complemented, and all can be tested for either state, not just one. Operating mode bits previously inaccessible may be read, tested, or saved. Situations where the 8051 instruction set provides new capabilities are contrasted with 8048 instruction sequences in Table 4.b. Here the 8051 speed advantage ranges from 5x to 15x!



Table 4b. (Continued)

| 8048 Instructions   | Bytes | Cycles | uSec | 8x51 Instructions | Bytes | Cycles & uSec |
|---|-------|--------|------|-------------------|-------|---------------|
| Turn Off Output Pin:<br>ANL PI.#0FBH = 2 2 5.0  |       |        |      | CLR PI.2 2 1      |       |               |
| Complement Output Pin:<br>IN A.PI<br>XRL A.#04H<br>OUTL PI.A = 4 6 15.0                           |       |        |      | CPL PI.2 2 1      |       |               |
| Clear Flag in RAM:<br>MOV R0.#FLGADR<br>MOV A.@R0<br>ANL A.#FLGMASK<br>MOV @R0.A = 6 6 15.0       |       |        |      | CLR USER_FLG 2 1  |       |               |
| Flag Testing<br>Jump if Software Flag is 0:<br>JF0 \$+4<br>JMP offset = 4 4 10.0                  |       |        |      | JNB F0.rel 3 2    |       |               |
| Jump if Accumulator bit is 0:<br>CPL A<br>JB7 offset<br>CPL A = 4 4 10.0                          |       |        |      | JNB ACC.7.rel 3 2 |       |               |
| Peripheral Polling<br>Test if Input Pin is Grounded:<br>IN A.PI<br>CPL A<br>JB3 offset = 4 5 12.5 |       |        |      | JNB PI.3.rel 3 2  |       |               |
| Test if Interrupt Pin is High:<br>JN1 \$+4<br>JMP offset = 4 4 10.0                               |       |        |      | JB INT0.rel 3 2   |       |               |

Combining Boolean and byte-wide instructions can produce great synergy. An MCS-51™ based application will prove to be:

- simpler to write since the architecture correlates more closely with the problems being solved;
- easier to debug because more individual instructions have no unexpected or undesirable side-effects;
- more byte efficient due to direct bit addressing and program counter relative branching;
- faster running because fewer bytes of instruction need to be fetched and fewer conditional jumps are processed;
- lower cost because of the high level of system-integration within one component.

These rather unabashed claims of excellence shall not go unsubstantiated. The rest of this chapter examines less trivial tasks simplified by the Boolean processor. The first

three compare the 8051 with other microprocessors; the last two go into 8051-based system designs in much greater depth.

### Design Example #1 - Bit Permutation

First off, we'll use the bit-transfer instructions to permute a lengthy pattern of bits.

A steadily increasing number of data communication products use encoding methods to protect the security of sensitive information. By law, interstate financial transactions involving the Federal banking system must be transmitted using the Federal Information Processing *Data Encryption Standard* (DES).

Basically, the DES combines eight bytes of "plaintext" data (in binary, ASCII, or any other format) with a 56-bit "key", producing a 64-bit encrypted value for transmission. At the receiving end the same algorithm is applied to the incoming data using the same key, reproducing the original eight byte message. The algorithm used for these permutations is fixed; different user-defined keys ensure data privacy.



It is not the purpose of this note to describe the DES in any detail. Suffice it to say that encryption/decryption is a long, iterative process consisting of rotations, exclusive-OR operations, function table look-ups, and an extensive (and quite bizarre) sequence of bit permutation, packing, and unpacking steps. (For further details refer to the June 21, 1979 issue of *Electronics* magazine.) The bit manipulation steps are included, it is rumored, to impede a general purpose digital supercomputer trying to "break" the code. Any algorithm implementing the DES with previous generation microprocessors would spend virtually all of its time diddling bits.

The bit manipulation performed is typified by the Key Schedule Calculation represented in Figure 9. This step is repeated 16 times for each key used in the course of a transmission. In essence, a seven-byte, 56-bit "Shifted Key Buffer" is transformed into an eight-byte, "Permutation Buffer" without altering the shifted Key. The arrows in Figure 9 indicate a few of the translation steps. Only six bits of each byte of the Permutation Buffer are used; the two high-order bits of each byte are cleared. This means only 48 of the 56 Shifted Key Buffer bits are used in any one iteration.

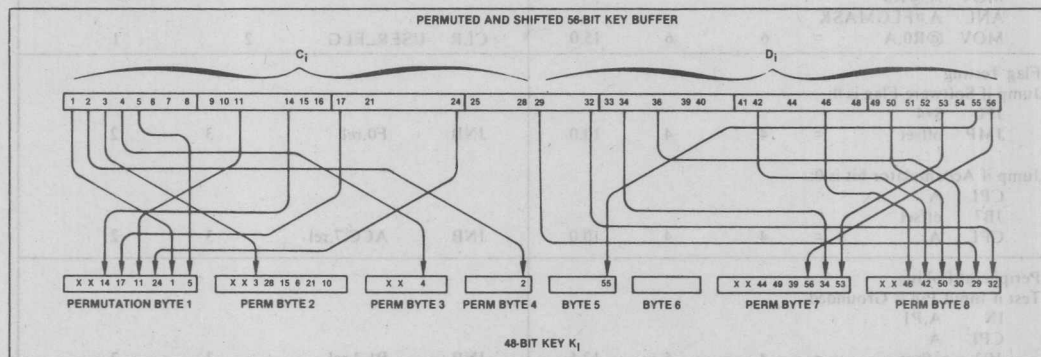


Figure 9. DES Key Schedule Transformation.

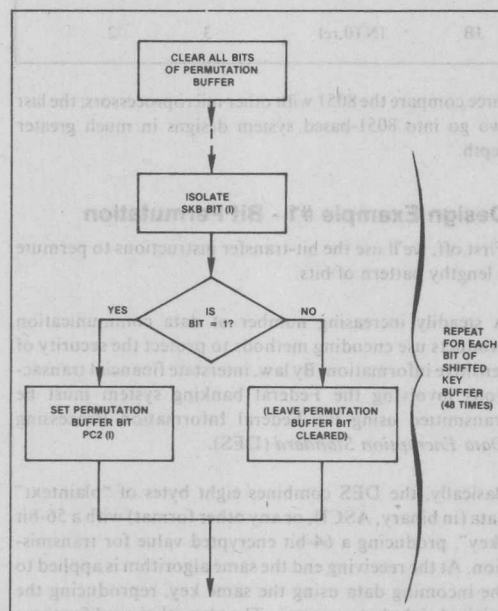


Figure 10.a. Flowchart for Key permutation attempted with a byte processor.

Different microprocessor architectures would best implement this type of permutation in different ways. Most approaches would share the steps of Figure 10.a:

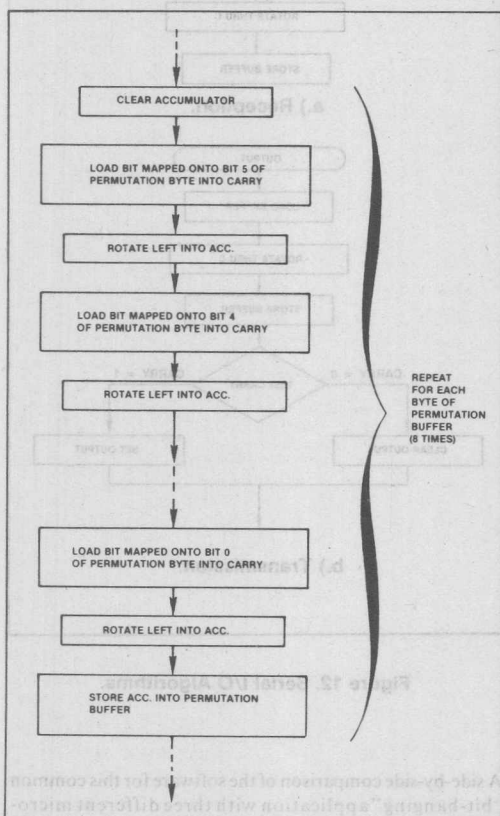
- Initialize the Permutation Buffer to default state (ones or zeroes);
- Isolate the state of a bit of a byte from the Key Buffer. Depending on the CPU, this might be accomplished by rotating a word of the Key Buffer through a carry flag or testing a bit in memory or an accumulator against a mask byte;
- Perform a conditional jump based on the carry or zero flag if the Permutation Buffer default state is correct;
- Otherwise reverse the corresponding bit in the permutation buffer with logical operations and mask bytes.

Each step above may require several instructions. The last three steps must be repeated for all 48 bits. Most microprocessors would spend 300 to 3,000 microseconds on each of the 16 iterations.

Notice, though, that this flow chart looks a lot like Figure 8. The Boolean Processor can permute bits by simply moving

them from the source to the carry to the destination—a total of two instructions taking four bytes and three microseconds per bit. Assume the Shifted Key Buffer and Permutation Buffer both reside in bit-addressable RAM, with the bits of the former assigned symbolic names SKB\_1, SKB\_2, . . . SKB\_56, and that the bytes of the latter are named PB\_1, . . . PB\_8. Then working from Figure 9, the software for the permutation algorithm would be that of Example 1.a. The total routine length would be 192 bytes, requiring 144 microseconds.

The algorithm of Figure 10.b is just slightly more efficient in this time-critical application and illustrates the synergy of an integrated byte and bit processor. The bits needed for each byte of the Permutation Buffer are assimilated by loading each bit into the carry (1 usec.) and shifting it into the accumulator (1 usec.). Each byte is stored in RAM when completed. Forty-eight bits thus need a total of 112 instructions, some of which are listed in Example 1.b.



**Figure 10.b. DES Key Permutation with Boolean Processor.**

Worst-case execution time would be 112 microseconds, since each instruction takes a single cycle. Routine length would also decrease, to 168 bytes. (Actually, in the context of the complete encryption algorithm, each permuted byte would be processed as soon as it is assimilated—saving memory and cutting execution time by another 8 usec.)

#### Example 1. DES Key Permutation Software.

##### a.) "Brute Force" technique.

```

MOV    C.SKB_1
MOV    PB_1.1.C
MOV    C.SKB_2
MOV    PB_4.0.C
MOV    C.SKB_3
MOV    PB_2.5.C
MOV    C.SKB_4
MOV    PB_1.0.C

```

```

...
MOV    C.SKB_55
MOV    PB_5.0.C
MOV    C.SKB_56
MOV    PB_7.2.C

```

##### b.) Using Accumulator to Collect Bits.

```

CLR    A
MOV    C.SKB_14
RLC    A
MOV    C.SKB_17
RLC    A
MOV    C.SKB_11
RLC    A
MOV    C.SKB_24
RLC    A
MOV    C.SKB_1
RLC    A
MOV    C.SKB_5
RLC    A
MOV    PB_1.A

```

```

MOV    C.SKB_29
RLC    A
MOV    C.SKB_32
RLC    A
MOV    PB_8.A

```

To date, most banking terminals and other systems using the DES have needed special boards or peripheral controller chips just for the encryption decryption process, and still more hardware to form a serial bit stream for transmission (Figure 11.a). An 8051 solution could pack most of the entire system onto the one chip (Figure 11.b). The whole DES algorithm would require less than one-

fourth of the on-chip program memory, with the remaining bytes free for operating the banking terminal (or whatever) itself.

Moreover, since transmission and reception of data is performed through the on-board UART, the unencrypted data (plaintext) never even exists outside the micro-computer! Naturally, this would afford a high degree of security from data interception.

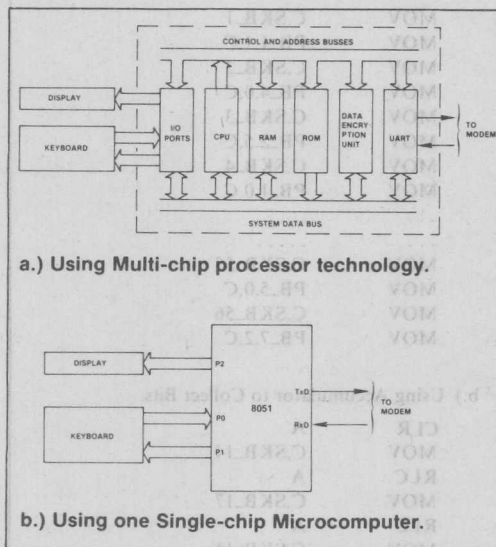


Figure 11. Secure Banking Terminal Block Diagram.

### Design Example #2 - Software Serial I/O

An exercise often imposed on beginning microcomputer students is to write a program simulating a UART. (See, for example, Application Notes AP24, AP29, and AP49.) Though doing this with the 8051 Family may appear to be a moot point (given that the hardware for a full UART is on-chip), it is still instructive to see how it would be done, and maintains a product line tradition.

As it turns out, the 8051 microcomputers can receive or transmit serial data via software very efficiently using the Boolean instruction set. Since any I/O pin may be a serial input or output, several serial links could be maintained at once.

Figures 12.a and 12.b show algorithms for receiving or transmitting a byte of data. (Another section of program would invoke this algorithm eight times, synchronizing it with a start bit, clock signal, software delay, or timer

interrupt.) Data is received by testing an input pin, setting the carry to the same state, shifting the carry into a data buffer, and saving the partial frame in internal RAM. Data is transmitted by shifting an output buffer through the carry, and generating each bit on an output pin.

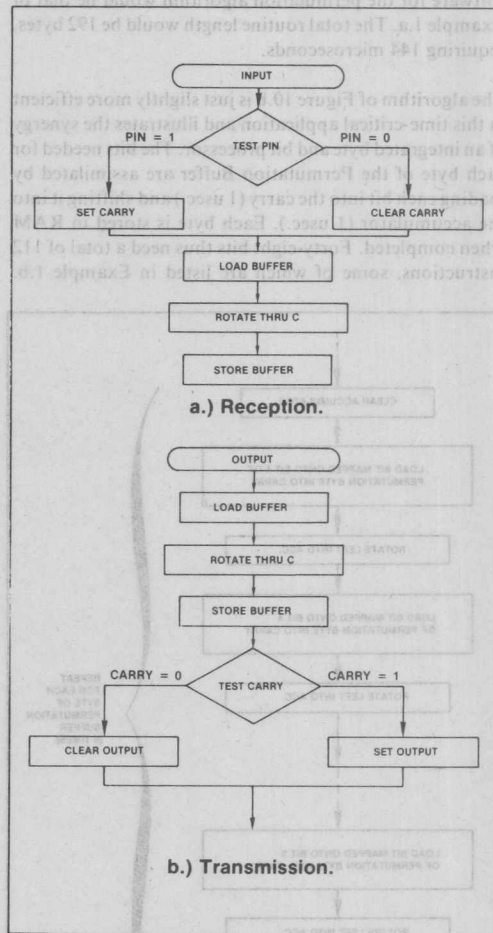


Figure 12. Serial I/O Algorithms.

A side-by-side comparison of the software for this common "bit-banging" application with three different micro-processor architectures is shown in Table 5.a and 5.b. The 8051 solution is more efficient than the others on every count!

Figure 13. Hardware Functions  
Table 5. Serial I/O Programs  
for Various Microprocessors.

|                     |                      |                |
|---------------------|----------------------|----------------|
| a.) Input Routine.  |                      |                |
| <b>8085</b>         | <b>8048</b>          | <b>8051</b>    |
| IN SERPORT          | CLR C                | MOV C,SERPIN   |
| ANI MASK            | JNT0 I.O             |                |
| JZ I.O              | CPL C                |                |
| CMC                 | MOV R0,#SERBUF       |                |
| I.O: LXI HL,SERBUF  | MOV A,R0             | MOV A,SERBUF   |
| MOV A,M             | RRC A                | RRC A          |
| RR                  | MOV @R0,A            | MOV SERBUF,A   |
| MOV M,A             |                      |                |
| RESULTS:            |                      |                |
| 8 INSTRUCTIONS      | 7 INSTRUCTIONS       | 4 INSTRUCTIONS |
| 14 BYTES            | 9 BYTES              | 7 BYTES        |
| 56 STATES           | 9 CYCLES             | 4 CYCLES       |
| 19 uSEC.            | 22.5 uSEC.           | 4 uSEC.        |
| b.) Output Routine. |                      |                |
| <b>8085</b>         | <b>8048</b>          | <b>8051</b>    |
| LXI HL,SERBUF       | MOV R0,#SERBUF       |                |
| MOV A,M             | MOV A,R0             | MOV A,SERBUF   |
| RR                  | RRC A                | RRC A          |
| MOV M,A             | MOV @R0,A            | MOV SERBUF,A   |
| IN SERPORT          |                      |                |
| JC HI               | JC HI                |                |
| I.O: ANI NOT MASK   | ANI SERPRT,#NOT MASK | MOV SERPIN,C   |
| JMP CNT             | JMP CNT              |                |
| HI: ORI MASK        | HI: ORI SERPRT,#MASK |                |
| CNT: OUT SERPORT    | CNT:                 |                |
| RESULTS:            |                      |                |
| 10 INSTRUCTIONS     | 8 INSTRUCTIONS       | 4 INSTRUCTIONS |
| 20 BYTES            | 13 BYTES             | 7 BYTES        |
| 72 STATES           | 11 CYCLES            | 5 CYCLES       |
| 24 uSEC.            | 27.5 uSEC.           | 5 uSEC.        |

### Design Example #3 - Combinatorial Logic Equations

Next we'll look at some simple uses for bit-test instructions and logical operations. (This example is also presented in Application Note AP-69.)

Virtually all hardware designers have solved complex functions using combinatorial logic. While the hardware involved may vary from relay logic, vacuum tubes, or TTL or to more esoteric technologies like fluidics, in each case the goal is the same: to solve a problem represented by a logical function of several Boolean variables.

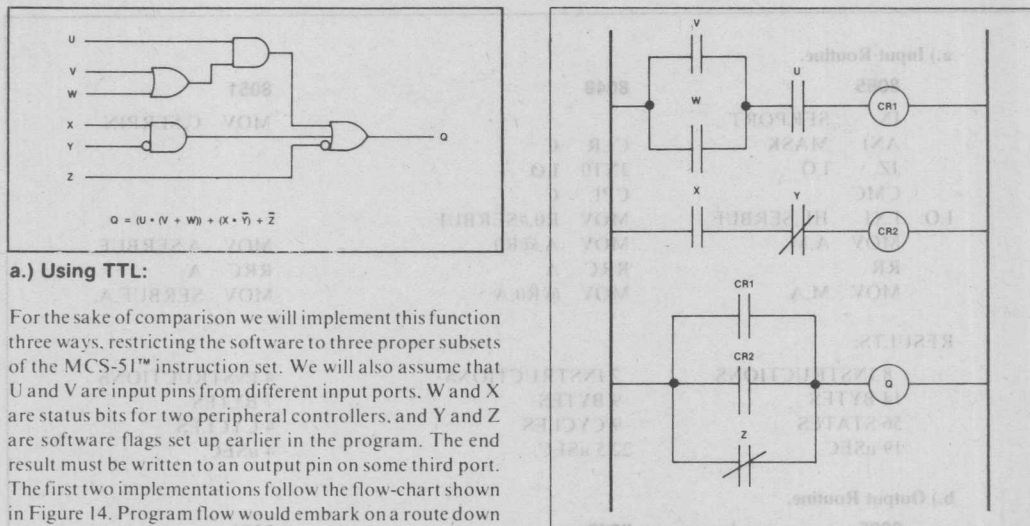
Figure 13 shows TTL and relay logic diagrams for a function of the six variables U through Z. Each is a solution of the equation.

$$Q = (U \cdot (V + W)) + (X \cdot \bar{Y}) + \bar{Z}$$

Equations of this sort might be reduced using Karnaugh Maps or algebraic techniques, but that is not the purpose of this example. As the logic complexity increases, so does the difficulty of the reduction process. Even a minor change to the function equations as the design evolves would require tedious re-reduction from scratch.



Figure 13. Hardware Implementations of Boolean functions.



#### a.) Using TTL:

For the sake of comparison we will implement this function three ways, restricting the software to three proper subsets of the MCS-51™ instruction set. We will also assume that U and V are input pins from different input ports. W and X are status bits for two peripheral controllers, and Y and Z are software flags set up earlier in the program. The end result must be written to an output pin on some third port. The first two implementations follow the flow-chart shown in Figure 14. Program flow would embark on a route down a test-and-branch tree and leaves either the "True" or "Not True" exit ASAP — as soon as the proper result has been determined. These exits then rewrite the output port with the result bit respectively one or zero.

Other digital computers must solve equations of this type with standard word-wide logical instructions and conditional jumps. So for the first implementation, we won't use any generalized bit-addressing instructions. As we shall soon see, being constrained to such an instruction subset produces somewhat sloppy software solutions. MCS-51™ mnemonics are used in Example 2.a; other machines might further cloud the situation by requiring operation-specific mnemonics like INPUT, OUTPUT, LOAD, STORE, etc., instead of the MOV mnemonic used for all variable transfers in the 8051 instruction set.

The code which results is cumbersome and error prone. It would be difficult to prove whether the software worked for all input combinations in programs of this sort. Furthermore, execution time will vary widely with input data.

Thanks to the direct bit-test operations, a single instruction can replace each move mask conditional jump sequence in Example 2.a, but the algorithm would be equally convoluted (see Example 2.B). To lessen the confusion "a bit" each input variable is assigned a symbolic name.

A more elegant and efficient implementation (Example 2.c) strings together the Boolean AND and ORL functions to generate the output function with straight-line code.

#### b.) Using Relay Logic:

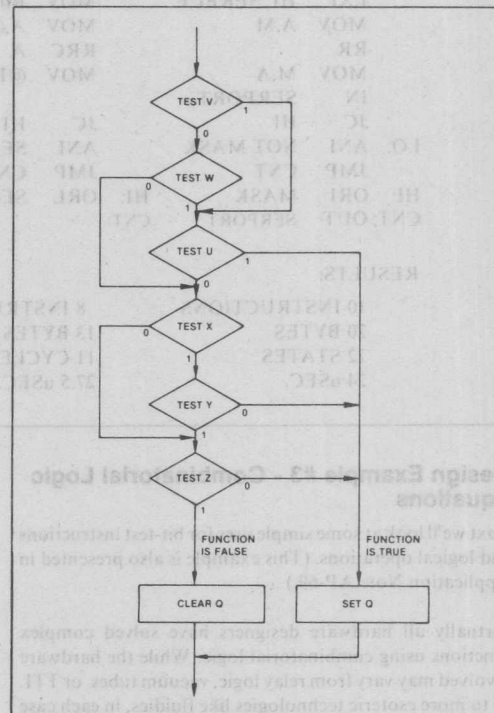


Figure 14. Flow chart for tree-branching algorithm.



When finished, the carry flag contains the result, which is simply copied out to the destination pin. No flow chart is needed—code can be written directly from the logic diagrams in Figure 14. The result is simplicity itself: fast, flexible, reliable, easy to design, and easy to debug.

An 8051 program can simulate an N-input AND or OR gate with at most N+1 lines of source program—one for each input and one line to store the results. To simulate NAND and NOR gates, complement the carry after computing the function. When some inputs to the gate have "inversion bubbles," perform the ANL or ORL operation on inverted operands. When the first input is inverted, either load the operand into the carry and then complement it, or use DeMorgan's Theorem to convert the gate to a different form.

Example 2. Software Solutions to Logic Function of Figure 13.

a.) Using only byte-wide logical instructions.

```
:BFUNC1 SOLVE RANDOM LOGIC FUNCTION
:      OF 6 VARIABLES BY LOADING AND
:      MASKING THE APPROPRIATE BITS
:      IN THE ACCUMULATOR, THEN
:      EXECUTING CONDITIONAL JUMPS
:      BASED ON ZERO CONDITION.
:      (APPROACH USED BY BYTE-
:      ORIENTED ARCHITECTURES.)
:      BYTE AND MASK VALUES
:      CORRESPOND TO RESPECTIVE BYTE
:      ADDRESS AND BIT POSITIONS.
```

OUTBUF DATA 22H :OUTPUT PIN STATE MAP

```
TESTV: MOV A,P2
:      ANL A,#00000100B
:      JNZ TESTU
:      MOV A,TCON
:      ANL A,#00100000B
:      JZ TESTX
TESTU: MOV A,P1
:      ANL A,#00000010B
:      JNZ SETQ
TESTX: MOV A,TCON
:      ANL A,#00001000B
:      JZ TESTZ
:      MOV A,20H
:      ANL A,#00000001B
:      JZ SETQ
TESTZ: MOV A,21H
:      ANL A,#00000010B
:      JZ SETQ
```

```
CLRQ: MOV A,OUTBUF
:      ANL A,#11110111B
:      JMP OUTQ
SETQ: MOV A,OUTBUF
:      ORL A,#00001000B
OUTQ: MOV OUTBUF,A
:      MOV P3,A
```

b.) Using only bit-test instructions.

```
:BFUNC2 SOLVE A RANDOM LOGIC FUNCTION
:      OF 6 VARIABLES BY DIRECTLY
:      POLLING EACH BIT.
:      (APPROACH USING MCS-51 UNIQUE
:      BIT-TEST INSTRUCTION CAPABILITY.)
:      SYMBOLS USED IN LOGIC DIAGRAM
:      ASSIGNED TO CORRESPONDING 8x51
:      BIT ADDRESSES.
```

```
:      BIT P1.1
V      BIT P2.2
W      BIT TF0
X      BIT IE1
Y      BIT 20H.0
Z      BIT 21H.1
Q      BIT P3.3
TEST_V: JB V,TEST_U
:      JNB W,TEST_X
TEST_U: JB U,SET_Q
TEST_X: JNB X,TEST_Z
:      JNB Y,SET_Q
TEST_Z: JNB Z,SET_Q
CLR_Q: CLR Q
JMP NXTTST
SET_Q: SETB Q
NXTTST: : (CONTINUATION OF
:      :PROGRAM)
```

c.) Using logical operations on Boolean variables.

```
:FUNC3 SOLVE A RANDOM LOGIC FUNCTION
:      OF 6 VARIABLES USING
:      STRAIGHTLINE LOGICAL
:      INSTRUCTIONS ON MCS-51 BOOLEAN
:      VARIABLES.
:      MOV C,V
:      ORL C,W :OUTPUT OF OR GATE
:      ANL C,U :OUTPUT OF TOP AND GATE
:      MOV F0,C :SAVE INTERMEDIATE STATE
:      MOV C,X
:      ANL C,Y :OUTPUT OF BOTTOM AND GATE
:      ORL C,F0 :INCLUDE VALUE SAVED ABOVE
:      ORL C,Z :INCLUDE LAST INPUT VARIABLE
:      MOV Q,C :OUTPUT COMPUTED RESULT
```

An upper-limit can be placed on the complexity of software to simulate a large number of gates by summing the total number of inputs and outputs. The *actual* total should be somewhat shorter, since calculations can be "chained," as shown above. The output of one gate is often the first input to another, bypassing the intermediate variable to eliminate two lines of source.

## Design Example #4 - Automotive Dashboard Functions

Now let's apply these techniques to designing the software for a complete controller system. This application is patterned after a familiar real-world application which isn't nearly as trivial as it might first appear: automobile turn signals.

Imagine the three position turn lever on the steering column as a single-pole, triple-throw toggle switch. In its central position all contacts are open. In the up or down positions contacts close causing corresponding lights in the rear of the car to blink. So far very simple.

Two more turn signals blink in the front of the car, and two others in the dashboard. All six bulbs flash when an emergency switch is closed. A thermo-mechanical relay (accessible under the dashboard in case it wears out) causes the blinking.

Applying the brake pedal turns the tail light filaments on constantly . . . unless a turn is in progress, in which case the blinking tail light is not affected. (Of course, the front turn signals and dashboard indicators are not affected by the brake pedal.) Table 6 summarizes these operating modes.

But we're not done yet. Each of the exterior turn signal (but not the dashboard) bulbs has a second, somewhat dimmer filament for the parking lights. Figure 15 shows TTL circuitry which could control all six bulbs. The signals labeled "High Freq." and "Low Freq." represent two square-wave inputs. Basically, when one of the turn switches is closed or the emergency switch is activated the low frequency signal (about 1 Hz) is gated through to the appropriate dashboard indicator(s) and turn signal(s). The rear signals are also activated when the brake pedal is depressed provided a turn is not being made in the same direction. When the parking light switch is closed the higher frequency oscillator is gated to each front and rear turn signal, sustaining a low-intensity background level. (This is to eliminate the need for additional parking light filaments.)

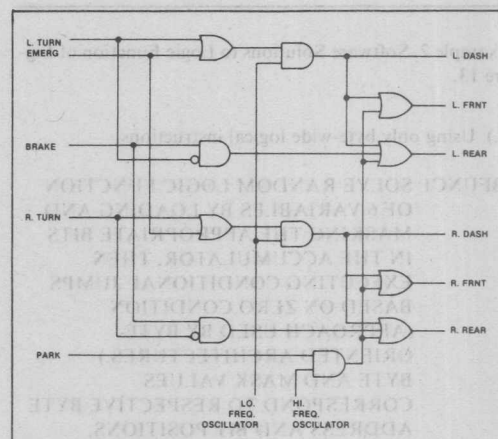


Figure 15. TTL logic implementation of automotive turn signals.

Table 6. Truth table for turn-signal operation.

| INPUT SIGNALS |               |                  |                   | OUTPUT SIGNALS    |                    |           |            |
|---------------|---------------|------------------|-------------------|-------------------|--------------------|-----------|------------|
| BRAKE SWITCH  | EMERG. SWITCH | LEFT TURN SWITCH | RIGHT TURN SWITCH | LEFT FRONT & DASH | RIGHT FRONT & DASH | LEFT REAR | RIGHT REAR |
| 0             | 0             | 0                | 0                 | OFF               | OFF                | OFF       | OFF        |
| 0             | 0             | 0                | 1                 | OFF               | BLINK              | OFF       | BLINK      |
| 0             | 0             | 1                | 0                 | BLINK             | OFF                | BLINK     | OFF        |
| 0             | 1             | 0                | 0                 | BLINK             | BLINK              | BLINK     | BLINK      |
| 0             | 1             | 0                | 1                 | BLINK             | BLINK              | BLINK     | BLINK      |
| 0             | 1             | 1                | 0                 | BLINK             | BLINK              | BLINK     | BLINK      |
| 1             | 0             | 0                | 0                 | OFF               | OFF                | ON        | ON         |
| 1             | 0             | 0                | 1                 | OFF               | BLINK              | ON        | BLINK      |
| 1             | 0             | 1                | 0                 | BLINK             | OFF                | BLINK     | ON         |
| 1             | 1             | 0                | 0                 | BLINK             | BLINK              | ON        | ON         |
| 1             | 1             | 0                | 1                 | BLINK             | BLINK              | ON        | BLINK      |
| 1             | 1             | 1                | 0                 | BLINK             | BLINK              | BLINK     | ON         |

In most cars, the switching logic to generate these functions requires a number of multiple-throw contacts. As many as 18 conductors thread the steering column of some automobiles solely for turn-signal and emergency blinker functions. (The author discovered this recently to his astonishment and dismay when replacing the whole assembly because of one burned contact.)

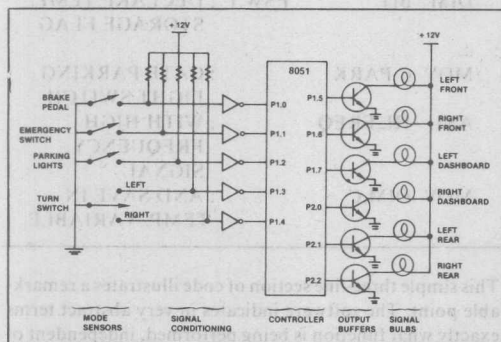
A multiple-conductor wiring harness runs to each corner of the car, behind the dash, up the steering column, and down to the blinker relay below. Connectors at each termination for each filament lead to extra cost and labor during construction, lower reliability and safety, and more costly repairs. And considering the system's present complexity, increasing its reliability or detecting failures would be quite difficult.

There are two reasons for going into such painful detail describing this example. First, to show that the messiest part of many system designs is determining what the controller should do. Writing the software to solve these functions will be comparatively easy. Secondly, to show the many potential failure points in the system. Later we'll see how the peripheral functions and intelligence built into a microcomputer (with a little creativity) can greatly reduce external interconnections and mechanical part count.

### The Single-chip Solution

The circuit shown in Figure 16 indicates five input pins to the five input variables—left-turn select, right-turn select, brake pedal down, emergency switch on, and parking lights on. Six output pins turn on the front, rear, and dashboard indicators for each side. The microcomputer implements all logical functions through software, which periodically updates the output signals as time elapses and input conditions change.

Figure 16. Microcomputer Turn-signal Connections.



Design Example #3 demonstrated that symbolic addressing with user-defined bit names makes code and documentation easier to write and maintain. Accordingly, we'll assign these I/O pins names for use throughout the program. (The format of this example will differ somewhat from the others. Segments of the overall program will be presented in sequence as each is described.)

```

:
:      INPUT PIN DECLARATIONS:
:      (ALL INPUTS ARE POSITIVE-TRUE LOGIC)
:
BRAKE  BIT P1.0 : BRAKE PEDAL DEPRESSED
EMERG  BIT P1.1 : EMERGENCY BLINKER
              ACTIVATED
PARK   BIT P1.2 : PARKING LIGHTS ON
L_TURN BIT P1.3 : TURN LEVER DOWN
R_TURN BIT P1.4 : TURN LEVER UP
:
:      OUTPUT PIN DECLARATIONS:
:
L_FRNT BIT P1.5 : FRONT LEFT-TURN
              INDICATOR
R_FRNT BIT P1.6 : FRONT RIGHT-TURN
              INDICATOR
L_DASH BIT P1.7 : DASHBOARD LEFT-TURN
              INDICATOR
R_DASH BIT P2.0 : DASHBOARD RIGHT-TURN
              INDICATOR
L_REAR BIT P2.1 : REAR LEFT-TURN
              INDICATOR
R_REAR BIT P2.2 : REAR RIGHT-TURN
              INDICATOR
:

```

Another key advantage of symbolic addressing will appear further on in the design cycle. The locations of cable connectors, signal conditioning circuitry, voltage regulators, heat sinks, and the like all affect P.C. board layout. It's quite likely that the somewhat arbitrary pin assignment defined early in the software design cycle will prove to be less than optimum; rearranging the I/O pin assignment could well allow a more compact module, or eliminate costly jumpers on a single-sided board. (These considerations apply especially to automotive and other cost-sensitive applications needing single-chip controllers.) Since other architectures mask bytes or use "clever" algorithms to isolate bits by rotating them into the carry, re-routing an input signal (from bit 1 of port 1, for example, to bit 4 of port 3) could require extensive modifications throughout the software.

The Boolean Processor's direct bit addressing makes such changes absolutely trivial. The number of the port containing the pin is irrelevant, and masks and complex program structures are not needed. Only the initial Boolean varia-

```

: INTERRUPT RATE SUBDIVIDER
SUB_DIV DATA 20H
: HIGH-FREQUENCY OSCILLATOR BIT
HL_FREQ BIT SUB_DIV.0
: LOW-FREQUENCY OSCILLATOR BIT
LO_FREQ BIT SUB_DIV.7
:
ORG 0000H
JMP INIT
:
ORG 100H
: PUT TIMER 0 IN MODE 1
INIT: MOV TMOD,#00000001B
: INITIALIZE TIMER REGISTERS
MOV TL0,#0
MOV TH0,#-16
: SUBDIVIDE INTERRUPT RATE BY 244
MOV SUB_DIV,#244
: ENABLE TIMER INTERRUPTS
SETB ET0
: GLOBALLY ENABLE ALL INTERRUPTS
SETB EA
: START TIMER
SETB TR0
: (CONTINUE WITH BACKGROUND PROGRAM)
: PUT TIMER 0 IN MODE 1
: INITIALIZE TIMER REGISTERS
: SUBDIVIDE INTERRUPT RATE BY 244
: ENABLE TIMER INTERRUPTS
: GLOBALLY ENABLE ALL INTERRUPTS
: START TIMER

```

ble declarations need to be changed; ASM51 automatically adjusts all addresses and symbolic references to the reassigned variables. The user is assured that no additional debugging or software verification will be required.

Timer 0 (one of the two on-chip timer counters) replaces the thermo-mechanical blinker relay in the dashboard controller. During system initialization it is configured as a timer in mode 1 by setting the least significant bit of the timer mode register (TMOD). In this configuration the low-order byte (TL0) is incremented every machine cycle, overflowing and incrementing the high-order byte (TH0) every 256  $\mu$ Sec. Timer interrupt 0 is enabled so that a hardware interrupt will occur each time TH0 overflows. (For details of the numerous timer operating modes see the MCS-51™ User's Manual.)

An eight-bit variable in the bit-addressable RAM array will be needed to further subdivide the interrupts via software. The lowest-order bit of this counter toggles very

fast to modulate the parking lights; bit 7 will be "tuned" to approximately 1 Hz for the turn- and emergency-indicator blinking rate.

Loading TH0 with -16 will cause an interrupt after 4.096 msec. The interrupt service routine reloads the high-order byte of timer 0 for the next interval, saves the CPU registers likely to be affected on the stack, and then decrements SUB\_DIV. Loading SUB\_DIV with 244 initially and each time it decrements to zero will produce a 0.999 second period for the highest-order bit.

```

ORG 000BH : TIMER 0 SERVICE VECTOR
MOV TH0,#-16
PUSH PSW
PUSH ACC
PUSH B
DJNZ SUB_DIV,T0SERV
MOV SUB_DIV,#244

```

The code to sample inputs, perform calculations, and update outputs—the real "meat" of the signal controller algorithm—may be performed either as part of the interrupt service routine or as part of a background program loop. The only concern is that it must be executed at least several dozen times per second to prevent parking light flickering. We will assume the former case, and insert the code into the timer 0 service routine.

First, notice from the logic diagram (Figure 15) that the subterm (PARK · H\_FREQ), asserted when the parking lights are to be on dimly, figures into four of the six output functions. Accordingly, we will first compute that term and save it in a temporary location named "DIM". The PSW contains two general purpose flags: F0, which corresponds to the 8048 flag of the same name, and PSW.1. Since The PSW has been saved and will be restored to its previous state after servicing the interrupt, we can use either bit for temporary storage.

```

DIM BIT PSW.1 : DECLARE TEMP.
                STORAGE FLAG
:
MOV C,PARK : GATE PARKING
: LIGHT SWITCH
ANI HL_FREQ : WITH HIGH
: FREQUENCY
: SIGNAL
MOV DIM,C : AND SAVE IN
: TEMP. VARIABLE.

```

This simple three-line section of code illustrates a remarkable point. The software indicates in very abstract terms exactly what function is being performed, independent of



the hardware configuration. The fact that these three bits include an input pin, a bit within a program variable, and a software flag in the PSW is totally invisible to the programmer.

Now generate and output the dashboard left turn signal.

```

MOV C.L_TURN      : SET CARRY IF
                   : TURN
ORI  C.EMERG       : OR EMERGENCY
                   : SELECTED.
ANI  C.I.O_FREQ    : GATE IN 1 HZ
                   : SIGNAL.
MOV  L_DASH.C      : AND OUTPUT TO
                   : DASHBOARD.

```

To generate the left front turn signal we only need to add the parking light function in F0. But notice that the function in the carry will also be needed for the rear signal. We can save effort later by saving its current state in F0.

```

MOV  F0.C          : SAVE FUNCTION
                   : SO FAR.
ORI  C.DIM         : ADD IN PARKING
                   : LIGHT FUNCTION
MOV  L_FRNT.C      : AND OUTPUT TO
                   : TURN SIGNAL.

```

Finally, the rear left turn signal should also be on when the brake pedal is depressed, provided a left turn is not in progress.

```

MOV  C.BRAKE       : GATE BRAKE
                   : PEDAL SWITCH
ANI  C.L_TURN       : WITH TURN
                   : LEVER.
ORI  C.F0          : INCLUDE TEMP.
                   : VARIABLE FROM
                   : DASH
ORI  C.DIM         : AND PARKING
                   : LIGHT FUNCTION
MOV  L_REAR.C      : AND OUTPUT TO
                   : TURN SIGNAL.

```

Now we have to go through a similar sequence for the right-hand equivalents to all the left-turn lights. This also gives us a chance to see how the code segments above look when combined.

```

MOV  C.R_TURN      : SET CARRY IF
                   : TURN
ORI  C.EMERG       : OR EMERGENCY
                   : SELECTED.
ANI  C.I.O_FREQ    : IF SO. GATE IN 1
                   : HZ SIGNAL.

```

```

MOV  R_DASH.C      : AND OUTPUT TO
                   : DASHBOARD.
MOV  F0.C          : SAVE FUNCTION
                   : SO FAR.
ORI  C.DIM         : ADD IN PARKING
                   : LIGHT FUNCTION
MOV  R_FRNT.C      : AND OUTPUT TO
                   : TURN SIGNAL.
MOV  C.BRAKE       : GATE BRAKE
                   : PEDAL SWITCH
ANI  C.R_TURN      : WITH TURN
                   : LEVER.
ORI  C.F0          : INCLUDE TEMP.
                   : VARIABLE FROM
                   : DASH
ORI  C.DIM         : AND PARKING
                   : LIGHT FUNCTION
MOV  R_REAR.C      : AND OUTPUT TO
                   : TURN SIGNAL.

```

(The perceptive reader may notice that simply rearranging the steps could eliminate one instruction from each sequence.)

Now that all six bulbs are in the proper states, we can return from the interrupt routine, and the program is finished. This code essentially needs to reverse the status saving steps at the beginning of the interrupt.

```

POP  B             : RESTORE CPU
                   : REGISTERS.
POP  ACC
POP  PSW
RETI

```

**Program Refinements:** The luminescence of an incandescent light bulb filament is generally non-linear; the 50% duty cycle of HL\_FREQ may not produce the desired intensity. If the application requires, duty cycles of 25%, 75%, etc. are easily achieved by ANDing and ORing in additional low-order bits of SUB\_DIV. For example, 30 Hz signals of seven different duty cycles could be produced by considering bits 2-0 as shown in Table 7. The only software change required would be to the code which sets-up variable DIM:

```

MOV  C.SUB_DIV.1   : START WITH 50
                   : PERCENT
ANI  C.SUB_DIV.0   : MASK DOWN TO 25
                   : PERCENT
ORI  C.SUB_DIV.2   : AND BUILD BACK TO
                   : 62 PERCENT
MOV  DIM.C         : DUTY CYCLE FOR
                   : PARKING LIGHTS.

```



Table 7. Non-trivial Duty Cycles.

| SUB_DIV BITS |   |   |   |   |   |   |   | DUTY CYCLES |       |       |       |       |       |       |
|--------------|---|---|---|---|---|---|---|-------------|-------|-------|-------|-------|-------|-------|
| 7            | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 12.5%       | 25.0% | 37.5% | 50.0% | 62.5% | 75.0% | 87.5% |
| X            | X | X | X | X | 0 | 0 | 0 | OFF         | OFF   | OFF   | OFF   | OFF   | OFF   | OFF   |
| X            | X | X | X | X | 0 | 0 | 1 | OFF         | OFF   | OFF   | OFF   | OFF   | OFF   | ON    |
| X            | X | X | X | X | 0 | 1 | 0 | OFF         | OFF   | OFF   | OFF   | OFF   | ON    | ON    |
| X            | X | X | X | X | 0 | 1 | 1 | OFF         | OFF   | OFF   | OFF   | ON    | ON    | ON    |
| X            | X | X | X | X | 1 | 0 | 0 | OFF         | OFF   | OFF   | ON    | ON    | ON    | ON    |
| X            | X | X | X | X | 1 | 0 | 1 | OFF         | OFF   | ON    | ON    | ON    | ON    | ON    |
| X            | X | X | X | X | 1 | 1 | 0 | OFF         | ON    | ON    | ON    | ON    | ON    | ON    |
| X            | X | X | X | X | 1 | 1 | 1 | ON          | ON    | ON    | ON    | ON    | ON    | ON    |

Interconnections increase cost and decrease reliability. The simple buffered pin-per-function circuit in Figure 16 is insufficient when many outputs require higher-than-TTL drive levels. A lower-cost solution uses the 8051 serial port in the shift-register mode to augment I/O. In mode 0, writing a byte to the serial port data buffer (SBUF) causes the data to be output sequentially through the "RXD" pin while a burst of eight clock pulses is generated on the "TXD" pin. A shift register connected to these pins (Figure 17) will load the data byte as it is shifted out. A number of special peripheral driver circuits combining shift-register inputs with high drive level outputs have been introduced recently.

Cascading multiple shift registers end-to-end will expand the number of outputs even further. The data rate in the I/O expansion mode is one megabaud, or 8 usec. per byte. This is the mode which the serial port defaults to following a reset, so no initialization is required.

The software for this technique uses the B register as a "map" corresponding to the different output functions. The program manipulates these bits instead of the output pins. After all functions have been calculated the B register is shifted by the serial port to the shift-register driver. (While some outputs may glitch as data is shifted through them, at 1 Megabaud most people wouldn't notice. Some shift registers provide an "enable" bit to hold the output states while new data is being shifted in.)

This is where the earlier decision to address bits symbolically throughout the program is going to pay off. This major I/O restructuring is nearly as simple to implement as rearranging the input pins. Again, only the bit declarations need to be changed.

```

L_FRNT BIT B.0 : FRONT LEFT-TURN
INDICATOR
R_FRNT BIT B.1 : FRONT RIGHT-TURN
INDICATOR
L_DASH BIT B.2 : DASHBOARD LEFT-TURN
INDICATOR
R_DASH BIT B.3 : DASHBOARD RIGHT-TURN
INDICATOR

```

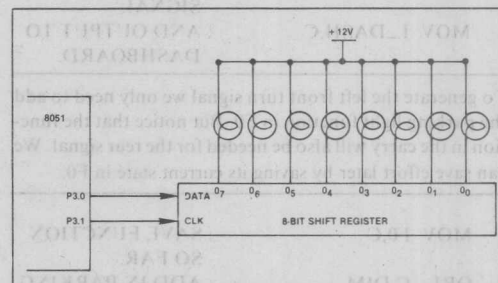


Figure 17. Output expansion using serial port.

```

L_REAR BIT B.4 : REAR LEFT-TURN
INDICATOR
R_REAR BIT B.5 : REAR RIGHT-TURN
INDICATOR

```

The original program to compute the functions need not change. After computing the output variables, the control map is transmitted to the buffered shift register through the serial port:

```
MOV SBUF,B : LOAD BUFFER AND TRANSMIT
```

The Boolean Processor solution holds a number of advantages over older methods. Fewer switches are required. Each is simpler, requiring fewer poles and lower current contacts. The flasher relay is eliminated entirely. Only six filaments are driven, rather than 10. The wiring harness is therefore simpler and less expensive—one conductor for each of the six lamps and each of the five sensor switches. The fewer conductors use far fewer connectors. The whole system is more reliable.

And since the system is much simpler it would be feasible to implement redundancy and/or fault detection on the four main turn indicators. Each could still be a standard double filament bulb, but with the filaments driven in parallel to tolerate single-element failures.

Even with redundancy, the lights will eventually fail. To handle this inescapable fact current or voltage sensing

circuits on each main drive wire can verify that each bulb and its high-current driver is functioning properly. Figure 18 shows one such circuit.

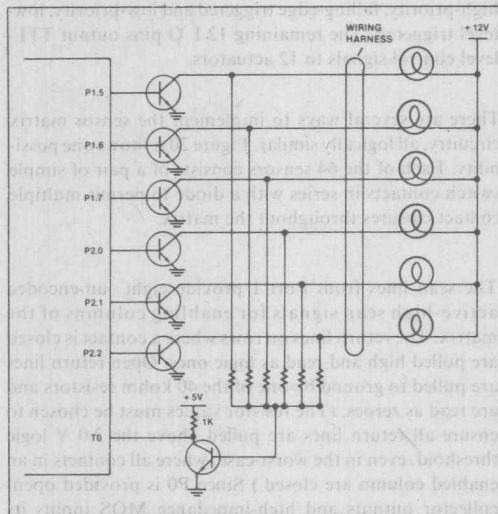


Figure 18.

Assume all of the lights are turned on except one; i.e., all but one of the collectors are grounded. For the bulb which is turned off, if there is continuity from +12 V through the bulb base and filament, the control wire, all connectors, and the P.C. board traces, and if the transistor is indeed not shorted to ground, then the collector will be pulled to +12 V. This turns on the base of Q8 through the corresponding resistor, and grounds the input pin, verifying that the bulb circuit is operational. The continuity of each circuit can be checked by software in this way.

Now turn *all* the bulbs on, grounding all the collectors. Q7 should be turned off, and the Test pin should be high. However, a control wire shorted to +12 V or an open-circuited drive transistor would leave one of the collectors at the higher voltage even now. This too would turn on Q7, indicating a different type of failure. Software could perform these checks once per second by executing the routine every time the software counter SUB\_DIV is reloaded by the interrupt routine.

```
DJNZ SUB_DIV,TOSERV
MOV SUB_DIV,#244      : RELOAD COUNTER
ORL P1,#11100000B    : SET CONTROL
                        : OUTPUTS HIGH
ORL P2,#00000111B
CLR I_FRNT            : FLOAT DRIVE
                        : COLLECTOR
JB T0,FAULT          : T0 SHOULD BE
                        : PULLED LOW
SETB I_FRNT           : PULL COLLECTOR
                        : BACK DOWN
```

```
CLR I_DASH
JB T0,FAULT
SETB I_DASH
CLR I_REAR
JB T0,FAULT
SETB I_REAR
CLR R_FRNT
JB T0,FAULT
SETB R_FRNT
CLR R_DASH
JB T0,FAULT
SETB R_DASH
CLR R_REAR
JB T0,FAULT
SETB R_REAR
```

```
: WITH ALL COLLECTORS GROUNDING, T0
: SHOULD BE HIGH
: IF SO, CONTINUE WITH INTERRUPT ROUTINE.
JB T0,TOSERV
FAULT:
: ELECTRICAL FAILURE
: PROCESSING ROUTINE
: (LEFT TO READER'S
: IMAGINATION)
TOSERV:
: CONTINUE WITH
: INTERRUPT PROCESSING
```

The complete assembled program listing is printed in Appendix A. The resulting code consists of 67 program statements, not counting declarations and comments, which assemble into 150 bytes of object code. Each pass through the service routine requires (coincidentally) 67 usec, plus 32 usec once per second for the electrical test. If executed every 4 msec as suggested this software would typically reduce the throughput of the background program by less than 2%.

Once a microcomputer has been designed into a system, new features suddenly become virtually free. Software could make the emergency blinkers flash alternately or at a rate faster than the turn signals. Turn signals could override the emergency blinkers. Adding more bulbs would allow multiple tail light sequencing and syncopation — true flash factor, so to speak.

## Design Example #5 - Complex Control Functions

Finally, we'll mix byte and bit operations to extend the use of 8051 into extremely complex applications.

Programmers can arbitrarily assign I/O pins to input and output functions only if the total does not exceed 32, which is insufficient for applications with a very large number of input variables. One way to expand the number of inputs is with a technique similar to multiplexed-keyboard scanning.

Figure 19 shows a block diagram for a moderately complex programmable industrial controller with the following characteristics:

- 64 input variable sensors;
- 12 output signals;
- Combinational and sequential logic computations;
- Remote operation with communications to a host processor via a high-speed full-duplex serial link;
- Two prioritized external interrupts;
- Internal real-time and time-of-day clocks.

While many microprocessors could be programmed to provide these capabilities with assorted peripheral support chips, an 8051 microcomputer needs **no** other integrated circuits!

The 64 input sensors are logically arranged as an 8x8 matrix. The pins of Port 1 sequentially enable each column of the sensor matrix; as each is enabled Port 0 reads in the state of each sensor in that column. An eight-byte block in bit-addressable RAM remembers the data as it is read in so that after each complete scan cycle there is an internal map of the current state of all sensors. Logic functions can then directly address the elements of the bit map.

The computer's serial port is configured as a nine-bit UART, transferring data at 17,000 bytes-per-second. The ninth bit may distinguish between address and data bytes.

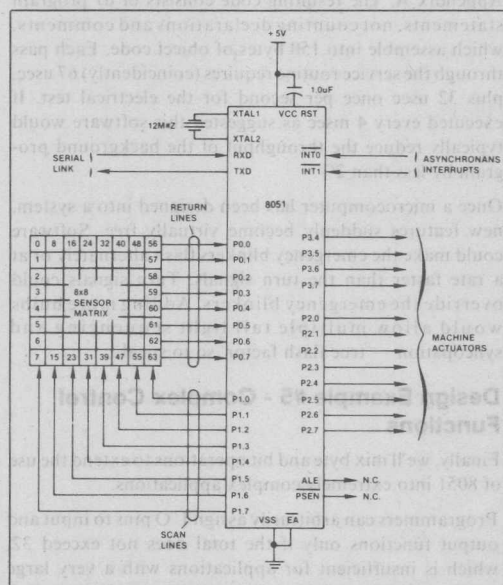


Figure 19. Block diagram of 64-input machine controller.

The 8051 serial port can be configured to detect bytes with the address bit set, automatically ignoring all others. Pins INT0 and INT1 are interrupts configured respectively as high-priority, falling-edge triggered and low-priority, low-level triggered. The remaining 12 I/O pins output TTL-level control signals to 12 actuators.

There are several ways to implement the sensor matrix circuitry, all logically similar. Figure 20.a shows one possibility. Each of the 64 sensors consists of a pair of simple switch contacts in series with a diode to permit multiple contact closures throughout the matrix.

The scan lines from Port 1 provide eight un-encoded active-high scan signals for enabling columns of the matrix. The return lines on rows where a contact is closed are pulled high and read as logic ones. Open return lines are pulled to ground by one of the 40 kohm resistors and are read as zeroes. (The resistor values must be chosen to ensure all return lines are pulled above the 2.0 V logic threshold, even in the worst-case, where all contacts in an enabled column are closed.) Since P0 is provided open-collector outputs and high-impedance MOS inputs its input loading may be considered negligible.

The circuits in Figures 20.b-20.d are variations on this theme. When input signals must be electrically isolated from the computer circuitry as in noisy industrial environments, phototransistors can replace the switch diode pairs and provide optical isolation as in Figure 20.b. Additional opto-isolators could also be used on the control output and special signal lines.

The other circuits assume that input signals are already at TTL levels. Figure 20.c uses octal three-state buffers enabled by active-low scan signals to gate eight signals onto Port 0. Port 0 is available for memory expansion or peripheral chip interfacing between sensor matrix scans. Eight-to-one multiplexers in Figure 20.d select one of eight inputs for each return line as determined by encoded address bits output on three pins of Port 1. (Five more output pins are thus freed for more control functions.) Each output can drive at least one standard TTL or up to 10 low-power TTL loads without additional buffering.

Going back to the original matrix circuit, Figure 21 shows the method used to scan the sensor matrix. Two complete bit maps are maintained in the bit-addressable region of the RAM: one for the current state and one for the previous state read for each sensor. If the need arises, the program could then sense input transitions and or debounce contact closures by comparing each bit with its earlier value.

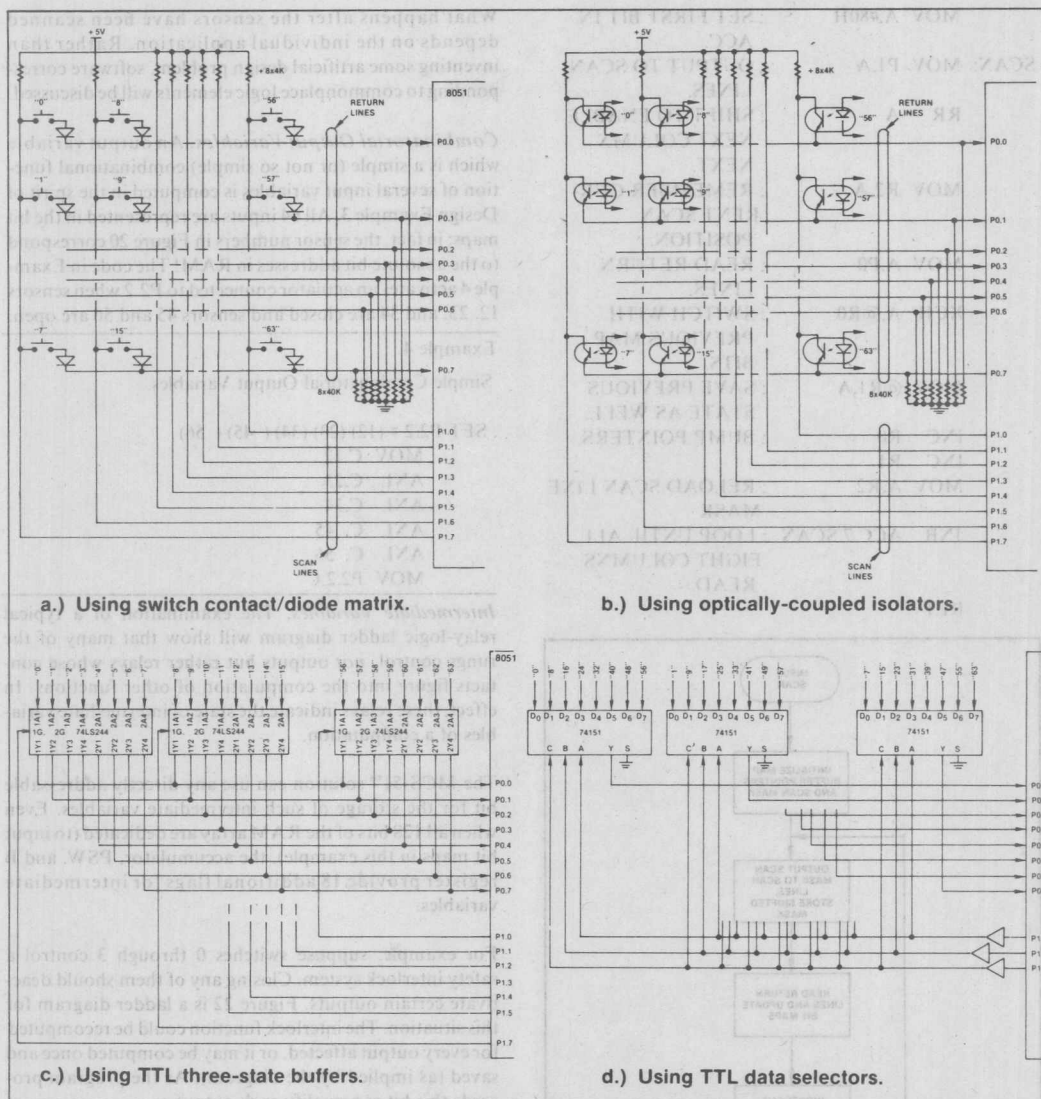


Figure 20. Sensor Matrix Implementation Methods.

The code in Example 3 implements the scanning algorithm for the circuits in Figure 20.a. Each column is enabled by setting a single bit in a field of zeroes. The bit maps are positive logic; ones represent contacts that are closed or isolators turned on.

Example 3.

```

INPUT_SCAN: SUBROUTINE TO READ
             CURRENT STATE
             OF 64 SENSORS AND
             SAVE IN RAM 20H-27H.
             MOV R0,#20H    INITIALIZE
             POINTERS
             MOV R1,#28H    FOR BIT MAP
             BASES.

```



```

MOV A,#80H      : SET FIRST BIT IN
                  ACC.
SCAN: MOV P1.A   : OUTPUT TO SCAN
                  LINES.
RR A            : SHIFT TO ENABLE
                  NEXT COLUMN
                  NEXT.
MOV R2.A        : REMEMBER CUR-
                  RENT SCAN
                  POSITION.
MOV A,P0        : READ RETURN
                  LINES.
XCH A,@R0       : SWITCH WITH
                  PREVIOUS MAP
                  BITS.
MOV @R1.A       : SAVE PREVIOUS
                  STATE AS WELL.
INC R0          : BUMP POINTERS.
INC R1
MOV A,R2        : RELOAD SCAN LINE
                  MASK
JNB ACC.7,SCAN  : LOOP UNTIL ALL
                  EIGHT COLUMNS
                  READ.

RET

```

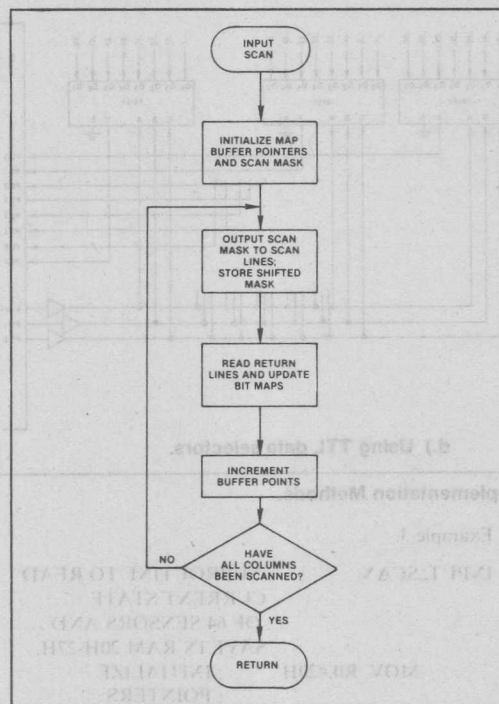


Figure 21. Flowchart for reading in sensor matrix.

What happens after the sensors have been scanned depends on the individual application. Rather than inventing some artificial design problem, software corresponding to commonplace logic elements will be discussed.

**Combinatorial Output Variables.** An output variable which is a simple (or not so simple) combinational function of several input variables is computed in the spirit of Design Example 3. All 64 inputs are represented in the bit maps; in fact, the sensor numbers in Figure 20 correspond to the absolute bit addresses in RAM! The code in Example 4 activates an actuator connected to P2.2 when sensors 12, 23, and 34 are closed and sensors 45 and 56 are open.

Example 4.

Simple Combinatorial Output Variables.

```

: SET P2.2 = (12) (23) (34) ( 45) ( 56)
MOV C,12
ANI C,23
ANI C,34
ANI C,45
ANI C,56
MOV P2.2,C

```

**Intermediate Variables.** The examination of a typical relay-logic ladder diagram will show that many of the rungs control *not* outputs but rather relays whose contacts figure into the computation of other functions. In effect, these relays indicate the state of intermediate variables of a computation.

The MCS-51™ solution can use any directly addressable bit for the storage of such intermediate variables. Even when all 128 bits of the RAM array are dedicated (to input bit maps in this example), the accumulator, PSW, and B register provide 18 additional flags for intermediate variables.

For example, suppose switches 0 through 3 control a safety interlock system. Closing any of them should deactivate certain outputs. Figure 22 is a ladder diagram for this situation. The interlock function could be recomputed for every output affected, or it may be computed once and saved (as implied by the diagram). As the program proceeds this bit can qualify each output.

Example 5. Incorporating Override signal into actuator outputs.

```

CALL INPUT_SCAN
MOV C,0
ORI C,1
ORI C,2
ORI C,3
MOV F0,C

```

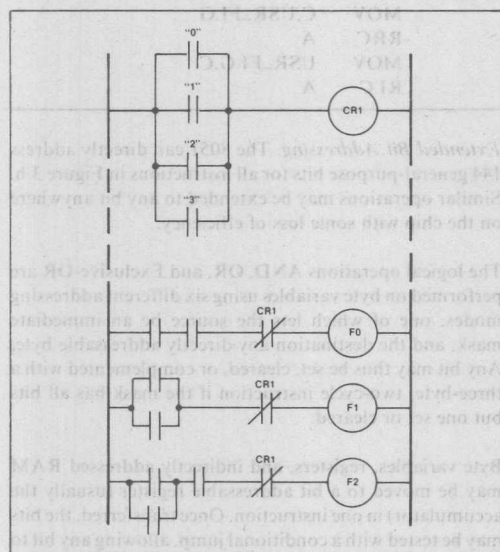
```

COMPUTE FUNCTION 0
ANL C, F0
MOV PI.0.C

COMPUTE FUNCTION 1
ANL C, F0
MOV PI.1.C

COMPUTE FUNCTION 2
ANL C, F0
MOV PI.2.C

```



**Figure 22. Ladder diagram for output override circuitry.**

**Latching Relays.** A latching relay can be forced into either the ON or OFF state by two corresponding input signals, where it will remain until forced onto the opposite state— analogous to a TTL Set/Reset flip-flop. The relay is used as an intermediate variable for other calculations. In the previous example, the emergency condition could be remembered and remain active until an “emergency cleared” button is pressed.

Any flag or addressable bit may represent a latching relay with a few lines of code (see Example 6).

**Example 6. Simulating a latching relay.**

```

I.SET: SET FLAG 0 IF C=1
I.SET: ORI C.F0
      MOV F0.C

I.RSET: RESET FLAG 0 IF C=1
I.RSET: CPS C
      ANL C.F0
      MOV F0.C

```

**Time Delay Relays.** A time delay relay does not respond to an input signal until it has been present (or absent) for some predefined time. For example, a ballast or load resistor may be switched in series with a D.C. motor when it is first turned on, and shunted from the circuit after one second. This sort of time delay may be simulated by an interrupt routine driven by one of the two 8051 timer counters. The procedure followed by the routine depends heavily on the details of the exact function needed: time-outs or time delays with resettable or non-resettable inputs are possible. If the interrupt routine is executed every 10 milliseconds the code in Example 7 will clear an intermediate variable set by the background program after it has been active for two seconds.

**Example 7. Code to clear USRFLAG after a fixed time delay.**

```

JNB USR_FLAG,NXTTST
DJNZ DIAY_COUNT,NXTTST
CLR USR_FLAG
MOV DIAY_COUNT,#200
NXTTST:

```

**Serial Interface to Remote Processor.** When it detects emergency conditions represented by certain input combinations (such as the earlier Emergency Override), the controller could shut down the machine immediately and/or alert the host processor via the serial port. Code bytes indicating the nature of the problem could be transmitted to a central computer. In fact, at 17,000 bytes-per-second, the entire contents of both bit maps could be sent to the host processor for further analysis in less than a millisecond! If the host decides that conditions warrant, it could alert other remote processors in the system that a problem exists and specify which shut-down sequence each should initiate. For more information on using the serial port, consult the MCS-51™ User's Manual.

**Response Timing.**

One difference between relay and programmed industrial controllers (when each is considered as a “black box”) is their respective reaction times to input changes. As reflected by a ladder diagram, relay systems contain a

large number of "rungs" operating in parallel. A change in input conditions will begin propagating through the system immediately, possibly affecting the output state within milliseconds.

Software, on the other hand, operates sequentially. A change in input states will not be detected until the next time an input scan is performed, and will not affect the outputs until that section of the program is reached. For that reason the raw speed of computing the logical functions is of extreme importance.

Here the Boolean processor pays off. Every instruction mentioned in this Note completes in one or two microseconds—the *minimum* instruction execution time for many other microcontrollers! A ladder diagram containing a hundred rungs, with an average of four contacts per rung can be replaced by approximately five hundred lines of software. A complete pass through the entire matrix scanning routine and all computations would require about a millisecond; less than the time it takes for most relays to change state.

A programmed controller which simulates each Boolean function with a subroutine would be less efficient by at least an order of magnitude. Extra software is needed for the simulation routines, and each step takes longer to execute for three reasons: several byte-wide logical instructions are executed per user program step (rather than one Boolean operation); most of those instructions take longer to execute with microprocessors performing multiple off-chip accesses; and calling and returning from the various subroutines requires overhead for stack operations.

In fact, the speed of the Boolean Processor solution is likely to be much faster than the system requires. The CPU might use the time left over to compute feedback parameters, collect and analyze execution statistics, perform system diagnostics, and so forth.

### Additional functions and uses.

With the building-block basics mentioned above many more operations may be synthesized by short instruction sequences.

**Exclusive-OR.** There are no common mechanical devices or relays analogous to the Exclusive-OR operation, so this instruction was omitted from the Boolean Processor. However, the Exclusive-OR or Exclusive-NOR operation may be performed in two instructions by conditionally complementing the carry or a Boolean variable based on the state of any other testable bit.

: EXCLUSIVE-OR FUNCTION IMPOSED ON CARRY  
: USING F0 IS INPUT VARIABLE.

```
XOR_F0: JNB  F0,XORCNT ;("JB" FOR X-NOR)
        CPL  C
```

```
XORCNT: ... ..
```

**XCH.** The contents of the carry and some other bit may be exchanged (switched) by using the accumulator as temporary storage. Bits can be moved into and out of the accumulator simultaneously using the Rotate-through-carry instructions, though this would alter the accumulator data.

: EXCHANGE CARRY WITH USRFLG

```
XCHBIT: RLC  A
        MOV  C,USR_FLG
        RRC  A
        MOV  USR_FLG,C
        RLC  A
```

**Extended Bit Addressing.** The 8051 can directly address 144 general-purpose bits for all instructions in Figure 3.b. Similar operations may be extended to any bit anywhere on the chip with some loss of efficiency.

The logical operations AND, OR, and Exclusive-OR are performed on byte variables using six different addressing modes, one of which lets the source be an immediate mask, and the destination any directly addressable byte. Any bit may thus be set, cleared, or complemented with a three-byte, two-cycle instruction if the mask has all bits but one set or cleared.

Byte variables, registers, and indirectly addressed RAM may be moved to a bit addressable register (usually the accumulator) in one instruction. Once transferred, the bits may be tested with a conditional jump, allowing any bit to be polled in 3 microseconds—still much faster than most architectures—or used for logical calculations. (This technique can also simulate additional bit addressing modes with byte operations.)

**Parity of bytes or bits.** The parity of the current accumulator contents is always available in the PSW, from whence it may be moved to the carry and further processed. Error-correcting Hamming codes and similar applications require computing parity on groups of isolated bits. This can be done by conditionally complementing the carry flag based on those bits or by gathering the bits into the accumulator (as shown in the DES example) and then testing the parallel parity flag.

**Multiple byte shift and CRC codes.**

Though the 8051 serial port can accommodate eight- or nine-bit data transmissions, some protocols involve much

longer bit streams. The algorithms presented in Design Example 2 can be extended quite readily to 16 or more bits by using multi-byte input and output buffers.

Many mass data storage peripherals and serial communications protocols include Cyclic Redundancy (CRC) codes to verify data integrity. The function is generally computed serially by hardware using shift registers and Exclusive-OR gates, but it can be done with software. As each bit is received into the carry, appropriate bits in the multi-byte data buffer are conditionally complemented based on the incoming data bit. When finished, the CRC register contents may be checked for zero by ORing the two bytes in the accumulator.

#### 4. SUMMARY

A truly unique facet of the Intel MCS-51™ microcomputer family design is the collection of features optimized for the one-bit operations so often desired in real-world, real-time control applications. Included are 17 special instructions, a Boolean accumulator, implicit and direct addressing modes, program and mass data storage, and many I/O options. These are the world's first single-chip microcomputers able to efficiently manipulate, operate on, and transfer either bytes or individual bits as data.

This Application Note has detailed the information needed by a microcomputer system designer to make full use of these capabilities. Five design examples were used to contrast the solutions allowed by the 8051 and those required by previous architectures. Depending on the individual application, the 8051 solution will be easier to design, more reliable to implement, debug, and verify, use less program memory, and run up to an order of magnitude faster than the same function implemented on previous digital computer architectures.

Combining byte- and bit-handling capabilities in a single microcomputer has a strong synergistic effect: the power of the result exceeds the power of byte- and bit-processors laboring individually. Virtually all user applications will benefit in some ways from this duality. Data intensive applications will use bit addressing for test pin monitoring or program control flags; control applications will use byte manipulation for parallel I/O expansion or arithmetic calculations.

It is hoped that these design examples give the reader an appreciation of these unique features and suggest ways to exploit them in his or her own application.

| ADDRESS | INSTRUCTION | OPERATION  | COMMENT                |
|---------|-------------|------------|------------------------|
| 0001    | MOV A, #0   | MOV A, #0  | INITIALIZE ACCUMULATOR |
| 0002    | MOV R0, #0  | MOV R0, #0 | INITIALIZE REGISTER R0 |
| 0003    | MOV R1, #0  | MOV R1, #0 | INITIALIZE REGISTER R1 |
| 0004    | MOV R2, #0  | MOV R2, #0 | INITIALIZE REGISTER R2 |
| 0005    | MOV R3, #0  | MOV R3, #0 | INITIALIZE REGISTER R3 |
| 0006    | MOV R4, #0  | MOV R4, #0 | INITIALIZE REGISTER R4 |
| 0007    | MOV R5, #0  | MOV R5, #0 | INITIALIZE REGISTER R5 |
| 0008    | MOV R6, #0  | MOV R6, #0 | INITIALIZE REGISTER R6 |
| 0009    | MOV R7, #0  | MOV R7, #0 | INITIALIZE REGISTER R7 |
| 000A    | MOV A, #0   | MOV A, #0  | INITIALIZE ACCUMULATOR |
| 000B    | MOV R0, #0  | MOV R0, #0 | INITIALIZE REGISTER R0 |
| 000C    | MOV R1, #0  | MOV R1, #0 | INITIALIZE REGISTER R1 |
| 000D    | MOV R2, #0  | MOV R2, #0 | INITIALIZE REGISTER R2 |
| 000E    | MOV R3, #0  | MOV R3, #0 | INITIALIZE REGISTER R3 |
| 000F    | MOV R4, #0  | MOV R4, #0 | INITIALIZE REGISTER R4 |
| 0010    | MOV R5, #0  | MOV R5, #0 | INITIALIZE REGISTER R5 |
| 0011    | MOV R6, #0  | MOV R6, #0 | INITIALIZE REGISTER R6 |
| 0012    | MOV R7, #0  | MOV R7, #0 | INITIALIZE REGISTER R7 |
| 0013    | MOV A, #0   | MOV A, #0  | INITIALIZE ACCUMULATOR |
| 0014    | MOV R0, #0  | MOV R0, #0 | INITIALIZE REGISTER R0 |
| 0015    | MOV R1, #0  | MOV R1, #0 | INITIALIZE REGISTER R1 |
| 0016    | MOV R2, #0  | MOV R2, #0 | INITIALIZE REGISTER R2 |
| 0017    | MOV R3, #0  | MOV R3, #0 | INITIALIZE REGISTER R3 |
| 0018    | MOV R4, #0  | MOV R4, #0 | INITIALIZE REGISTER R4 |
| 0019    | MOV R5, #0  | MOV R5, #0 | INITIALIZE REGISTER R5 |
| 001A    | MOV R6, #0  | MOV R6, #0 | INITIALIZE REGISTER R6 |
| 001B    | MOV R7, #0  | MOV R7, #0 | INITIALIZE REGISTER R7 |
| 001C    | MOV A, #0   | MOV A, #0  | INITIALIZE ACCUMULATOR |
| 001D    | MOV R0, #0  | MOV R0, #0 | INITIALIZE REGISTER R0 |
| 001E    | MOV R1, #0  | MOV R1, #0 | INITIALIZE REGISTER R1 |
| 001F    | MOV R2, #0  | MOV R2, #0 | INITIALIZE REGISTER R2 |
| 0020    | MOV R3, #0  | MOV R3, #0 | INITIALIZE REGISTER R3 |
| 0021    | MOV R4, #0  | MOV R4, #0 | INITIALIZE REGISTER R4 |
| 0022    | MOV R5, #0  | MOV R5, #0 | INITIALIZE REGISTER R5 |
| 0023    | MOV R6, #0  | MOV R6, #0 | INITIALIZE REGISTER R6 |
| 0024    | MOV R7, #0  | MOV R7, #0 | INITIALIZE REGISTER R7 |
| 0025    | MOV A, #0   | MOV A, #0  | INITIALIZE ACCUMULATOR |
| 0026    | MOV R0, #0  | MOV R0, #0 | INITIALIZE REGISTER R0 |
| 0027    | MOV R1, #0  | MOV R1, #0 | INITIALIZE REGISTER R1 |
| 0028    | MOV R2, #0  | MOV R2, #0 | INITIALIZE REGISTER R2 |
| 0029    | MOV R3, #0  | MOV R3, #0 | INITIALIZE REGISTER R3 |
| 002A    | MOV R4, #0  | MOV R4, #0 | INITIALIZE REGISTER R4 |
| 002B    | MOV R5, #0  | MOV R5, #0 | INITIALIZE REGISTER R5 |
| 002C    | MOV R6, #0  | MOV R6, #0 | INITIALIZE REGISTER R6 |
| 002D    | MOV R7, #0  | MOV R7, #0 | INITIALIZE REGISTER R7 |
| 002E    | MOV A, #0   | MOV A, #0  | INITIALIZE ACCUMULATOR |
| 002F    | MOV R0, #0  | MOV R0, #0 | INITIALIZE REGISTER R0 |
| 0030    | MOV R1, #0  | MOV R1, #0 | INITIALIZE REGISTER R1 |
| 0031    | MOV R2, #0  | MOV R2, #0 | INITIALIZE REGISTER R2 |
| 0032    | MOV R3, #0  | MOV R3, #0 | INITIALIZE REGISTER R3 |
| 0033    | MOV R4, #0  | MOV R4, #0 | INITIALIZE REGISTER R4 |
| 0034    | MOV R5, #0  | MOV R5, #0 | INITIALIZE REGISTER R5 |
| 0035    | MOV R6, #0  | MOV R6, #0 | INITIALIZE REGISTER R6 |
| 0036    | MOV R7, #0  | MOV R7, #0 | INITIALIZE REGISTER R7 |
| 0037    | MOV A, #0   | MOV A, #0  | INITIALIZE ACCUMULATOR |
| 0038    | MOV R0, #0  | MOV R0, #0 | INITIALIZE REGISTER R0 |
| 0039    | MOV R1, #0  | MOV R1, #0 | INITIALIZE REGISTER R1 |
| 003A    | MOV R2, #0  | MOV R2, #0 | INITIALIZE REGISTER R2 |
| 003B    | MOV R3, #0  | MOV R3, #0 | INITIALIZE REGISTER R3 |
| 003C    | MOV R4, #0  | MOV R4, #0 | INITIALIZE REGISTER R4 |
| 003D    | MOV R5, #0  | MOV R5, #0 | INITIALIZE REGISTER R5 |
| 003E    | MOV R6, #0  | MOV R6, #0 | INITIALIZE REGISTER R6 |
| 003F    | MOV R7, #0  | MOV R7, #0 | INITIALIZE REGISTER R7 |
| 0040    | MOV A, #0   | MOV A, #0  | INITIALIZE ACCUMULATOR |
| 0041    | MOV R0, #0  | MOV R0, #0 | INITIALIZE REGISTER R0 |
| 0042    | MOV R1, #0  | MOV R1, #0 | INITIALIZE REGISTER R1 |
| 0043    | MOV R2, #0  | MOV R2, #0 | INITIALIZE REGISTER R2 |
| 0044    | MOV R3, #0  | MOV R3, #0 | INITIALIZE REGISTER R3 |
| 0045    | MOV R4, #0  | MOV R4, #0 | INITIALIZE REGISTER R4 |
| 0046    | MOV R5, #0  | MOV R5, #0 | INITIALIZE REGISTER R5 |
| 0047    | MOV R6, #0  | MOV R6, #0 | INITIALIZE REGISTER R6 |
| 0048    | MOV R7, #0  | MOV R7, #0 | INITIALIZE REGISTER R7 |
| 0049    | MOV A, #0   | MOV A, #0  | INITIALIZE ACCUMULATOR |
| 004A    | MOV R0, #0  | MOV R0, #0 | INITIALIZE REGISTER R0 |
| 004B    | MOV R1, #0  | MOV R1, #0 | INITIALIZE REGISTER R1 |
| 004C    | MOV R2, #0  | MOV R2, #0 | INITIALIZE REGISTER R2 |
| 004D    | MOV R3, #0  | MOV R3, #0 | INITIALIZE REGISTER R3 |
| 004E    | MOV R4, #0  | MOV R4, #0 | INITIALIZE REGISTER R4 |
| 004F    | MOV R5, #0  | MOV R5, #0 | INITIALIZE REGISTER R5 |
| 0050    | MOV R6, #0  | MOV R6, #0 | INITIALIZE REGISTER R6 |
| 0051    | MOV R7, #0  | MOV R7, #0 | INITIALIZE REGISTER R7 |
| 0052    | MOV A, #0   | MOV A, #0  | INITIALIZE ACCUMULATOR |
| 0053    | MOV R0, #0  | MOV R0, #0 | INITIALIZE REGISTER R0 |
| 0054    | MOV R1, #0  | MOV R1, #0 | INITIALIZE REGISTER R1 |
| 0055    | MOV R2, #0  | MOV R2, #0 | INITIALIZE REGISTER R2 |
| 0056    | MOV R3, #0  | MOV R3, #0 | INITIALIZE REGISTER R3 |
| 0057    | MOV R4, #0  | MOV R4, #0 | INITIALIZE REGISTER R4 |
| 0058    | MOV R5, #0  | MOV R5, #0 | INITIALIZE REGISTER R5 |
| 0059    | MOV R6, #0  | MOV R6, #0 | INITIALIZE REGISTER R6 |
| 005A    | MOV R7, #0  | MOV R7, #0 | INITIALIZE REGISTER R7 |
| 005B    | MOV A, #0   | MOV A, #0  | INITIALIZE ACCUMULATOR |
| 005C    | MOV R0, #0  | MOV R0, #0 | INITIALIZE REGISTER R0 |
| 005D    | MOV R1, #0  | MOV R1, #0 | INITIALIZE REGISTER R1 |
| 005E    | MOV R2, #0  | MOV R2, #0 | INITIALIZE REGISTER R2 |
| 005F    | MOV R3, #0  | MOV R3, #0 | INITIALIZE REGISTER R3 |
| 0060    | MOV R4, #0  | MOV R4, #0 | INITIALIZE REGISTER R4 |
| 0061    | MOV R5, #0  | MOV R5, #0 | INITIALIZE REGISTER R5 |
| 0062    | MOV R6, #0  | MOV R6, #0 | INITIALIZE REGISTER R6 |
| 0063    | MOV R7, #0  | MOV R7, #0 | INITIALIZE REGISTER R7 |
| 0064    | MOV A, #0   | MOV A, #0  | INITIALIZE ACCUMULATOR |
| 0065    | MOV R0, #0  | MOV R0, #0 | INITIALIZE REGISTER R0 |
| 0066    | MOV R1, #0  | MOV R1, #0 | INITIALIZE REGISTER R1 |
| 0067    | MOV R2, #0  | MOV R2, #0 | INITIALIZE REGISTER R2 |
| 0068    | MOV R3, #0  | MOV R3, #0 | INITIALIZE REGISTER R3 |
| 0069    | MOV R4, #0  | MOV R4, #0 | INITIALIZE REGISTER R4 |
| 006A    | MOV R5, #0  | MOV R5, #0 | INITIALIZE REGISTER R5 |
| 006B    | MOV R6, #0  | MOV R6, #0 | INITIALIZE REGISTER R6 |
| 006C    | MOV R7, #0  | MOV R7, #0 | INITIALIZE REGISTER R7 |
| 006D    | MOV A, #0   | MOV A, #0  | INITIALIZE ACCUMULATOR |
| 006E    | MOV R0, #0  | MOV R0, #0 | INITIALIZE REGISTER R0 |
| 006F    | MOV R1, #0  | MOV R1, #0 | INITIALIZE REGISTER R1 |
| 0070    | MOV R2, #0  | MOV R2, #0 | INITIALIZE REGISTER R2 |
| 0071    | MOV R3, #0  | MOV R3, #0 | INITIALIZE REGISTER R3 |
| 0072    | MOV R4, #0  | MOV R4, #0 | INITIALIZE REGISTER R4 |
| 0073    | MOV R5, #0  | MOV R5, #0 | INITIALIZE REGISTER R5 |
| 0074    | MOV R6, #0  | MOV R6, #0 | INITIALIZE REGISTER R6 |
| 0075    | MOV R7, #0  | MOV R7, #0 | INITIALIZE REGISTER R7 |
| 0076    | MOV A, #0   | MOV A, #0  | INITIALIZE ACCUMULATOR |
| 0077    | MOV R0, #0  | MOV R0, #0 | INITIALIZE REGISTER R0 |
| 0078    | MOV R1, #0  | MOV R1, #0 | INITIALIZE REGISTER R1 |
| 0079    | MOV R2, #0  | MOV R2, #0 | INITIALIZE REGISTER R2 |
| 007A    | MOV R3, #0  | MOV R3, #0 | INITIALIZE REGISTER R3 |
| 007B    | MOV R4, #0  | MOV R4, #0 | INITIALIZE REGISTER R4 |
| 007C    | MOV R5, #0  | MOV R5, #0 | INITIALIZE REGISTER R5 |
| 007D    | MOV R6, #0  | MOV R6, #0 | INITIALIZE REGISTER R6 |
| 007E    | MOV R7, #0  | MOV R7, #0 | INITIALIZE REGISTER R7 |
| 007F    | MOV A, #0   | MOV A, #0  | INITIALIZE ACCUMULATOR |
| 0080    | MOV R0, #0  | MOV R0, #0 | INITIALIZE REGISTER R0 |
| 0081    | MOV R1, #0  | MOV R1, #0 | INITIALIZE REGISTER R1 |
| 0082    | MOV R2, #0  | MOV R2, #0 | INITIALIZE REGISTER R2 |
| 0083    | MOV R3, #0  | MOV R3, #0 | INITIALIZE REGISTER R3 |
| 0084    | MOV R4, #0  | MOV R4, #0 | INITIALIZE REGISTER R4 |
| 0085    | MOV R5, #0  | MOV R5, #0 | INITIALIZE REGISTER R5 |
| 0086    | MOV R6, #0  | MOV R6, #0 | INITIALIZE REGISTER R6 |
| 0087    | MOV R7, #0  | MOV R7, #0 | INITIALIZE REGISTER R7 |
| 0088    | MOV A, #0   | MOV A, #0  | INITIALIZE ACCUMULATOR |
| 0089    | MOV R0, #0  | MOV R0, #0 | INITIALIZE REGISTER R0 |
| 008A    | MOV R1, #0  | MOV R1, #0 | INITIALIZE REGISTER R1 |
| 008B    | MOV R2, #0  | MOV R2, #0 | INITIALIZE REGISTER R2 |
| 008C    | MOV R3, #0  | MOV R3, #0 | INITIALIZE REGISTER R3 |
| 008D    | MOV R4, #0  | MOV R4, #0 | INITIALIZE REGISTER R4 |
| 008E    | MOV R5, #0  | MOV R5, #0 | INITIALIZE REGISTER R5 |
| 008F    | MOV R6, #0  | MOV R6, #0 | INITIALIZE REGISTER R6 |
| 0090    | MOV R7, #0  | MOV R7, #0 | INITIALIZE REGISTER R7 |
| 0091    | MOV A, #0   | MOV A, #0  | INITIALIZE ACCUMULATOR |
| 0092    | MOV R0, #0  | MOV R0, #0 | INITIALIZE REGISTER R0 |
| 0093    | MOV R1, #0  | MOV R1, #0 | INITIALIZE REGISTER R1 |
| 0094    | MOV R2, #0  | MOV R2, #0 | INITIALIZE REGISTER R2 |
| 0095    | MOV R3, #0  | MOV R3, #0 | INITIALIZE REGISTER R3 |
| 0096    | MOV R4, #0  | MOV R4, #0 | INITIALIZE REGISTER R4 |
| 0097    | MOV R5, #0  | MOV R5, #0 | INITIALIZE REGISTER R5 |
| 0098    | MOV R6, #0  | MOV R6, #0 | INITIALIZE REGISTER R6 |
| 0099    | MOV R7, #0  | MOV R7, #0 | INITIALIZE REGISTER R7 |
| 009A    | MOV A, #0   | MOV A, #0  | INITIALIZE ACCUMULATOR |
| 009B    | MOV R0, #0  | MOV R0, #0 | INITIALIZE REGISTER R0 |
| 009C    | MOV R1, #0  | MOV R1, #0 | INITIALIZE REGISTER R1 |
| 009D    | MOV R2, #0  | MOV R2, #0 | INITIALIZE REGISTER R2 |
| 009E    | MOV R3, #0  | MOV R3, #0 | INITIALIZE REGISTER R3 |
| 009F    | MOV R4, #0  | MOV R4, #0 | INITIALIZE REGISTER R4 |
| 00A0    | MOV R5, #0  | MOV R5, #0 | INITIALIZE REGISTER R5 |
| 00A1    | MOV R6, #0  | MOV R6, #0 | INITIALIZE REGISTER R6 |
| 00A2    | MOV R7, #0  | MOV R7, #0 | INITIALIZE REGISTER R7 |
| 00A3    | MOV A, #0   | MOV A, #0  | INITIALIZE ACCUMULATOR |
| 00A4    | MOV R0, #0  | MOV R0, #0 | INITIALIZE REGISTER R0 |
| 00A5    | MOV R1, #0  | MOV R1, #0 | INITIALIZE REGISTER R1 |
| 00A6    | MOV R2, #0  | MOV R2, #0 | INITIALIZE REGISTER R2 |
| 00A7    | MOV R3, #0  | MOV R3, #0 | INITIALIZE REGISTER R3 |
| 00A8    | MOV R4, #0  | MOV R4, #0 | INITIALIZE REGISTER R4 |
| 00A9    | MOV R5, #0  | MOV R5, #0 | INITIALIZE REGISTER R5 |
| 00AA    | MOV R6, #0  | MOV R6, #0 | INITIALIZE REGISTER R6 |
| 00AB    | MOV R7, #0  | MOV R7, #0 | INITIALIZE REGISTER R7 |
| 00AC    | MOV A, #0   | MOV A, #0  | INITIALIZE ACCUMULATOR |
| 00AD    | MOV R0, #0  | MOV R0, #0 | INITIALIZE REGISTER R0 |
| 00AE    | MOV R1, #0  | MOV R1, #0 | INITIALIZE REGISTER R1 |
| 00AF    | MOV R2, #0  | MOV R2, #0 | INITIALIZE REGISTER R2 |
| 00B0    | MOV R3, #0  | MOV R3, #0 | INITIALIZE REGISTER R3 |
| 00B1    | MOV R4, #0  | MOV R4, #0 | INITIALIZE REGISTER R4 |
| 00B2    | MOV R5, #0  | MOV R5, #0 | INITIALIZE REGISTER R5 |
| 00B3    | MOV R6, #0  | MOV R6, #0 | INITIALIZE REGISTER R6 |
| 00B4    | MOV R7, #0  | MOV R7, #0 | INITIALIZE REGISTER R7 |
| 00B5    | MOV A, #0   | MOV A, #0  | INITIALIZE ACCUMULATOR |
| 00B6    | MOV R0, #0  | MOV R0, #0 | INITIALIZE REGISTER R0 |
| 00B7    | MOV R1, #0  | MOV R1, #0 | INITIALIZE REGISTER R1 |
| 00B8    | MOV R2, #0  | MOV R2, #0 | INITIALIZE REGISTER R2 |
| 00B9    | MOV R3, #0  | MOV R3, #0 | INITIALIZE REGISTER R3 |
| 00BA    | MOV R4, #0  | MOV R4, #0 | INITIALIZE REGISTER R4 |
| 00BB    | MOV R5, #0  | MOV R5, #0 | INITIALIZE REGISTER R5 |
| 00BC    | MOV R6, #0  | MOV R6, #0 | INITIALIZE REGISTER R6 |
| 00BD    | MOV R7, #0  | MOV R7, #0 | INITIALIZE REGISTER R7 |
| 00BE    | MOV A, #0   | MOV A, #0  | INITIALIZE ACCUMULATOR |
| 00BF    | MOV R0, #0  | MOV R0, #0 | INITIALIZE REGISTER R0 |
| 00C0    | MOV R1, #0  | MOV R1, #0 | INITIALIZE REGISTER R1 |
| 00C1    | MOV R2, #0  | MOV R2, #0 | INITIALIZE REGISTER R2 |
| 00C2    | MOV R3, #0  | MOV R3, #0 | INITIALIZE REGISTER R3 |
| 00C3    | MOV R4, #0  | MOV R4, #0 | INITIALIZE REGISTER R4 |
| 00C4    | MOV R5, #0  | MOV R5, #0 | INITIALIZE REGISTER R5 |
| 00C5    | MOV R6, #0  | MOV R6, #0 | INITIALIZE REGISTER R6 |
| 00C6    | MOV R7, #0  | MOV R7, #0 | INITIALIZE REGISTER R7 |
| 00C7    | MOV A, #0   | MOV A, #0  | INITIALIZE ACCUMULATOR |
| 00C8    | MOV R0, #0  | MOV R0, #0 | INITIALIZE REGISTER R0 |
| 00C9    | MOV R1, #0  | MOV R1, #0 | INITIALIZE REGISTER R1 |
| 00CA    | MOV R2, #0  | MOV R2, #0 | INITIALIZE REGISTER R2 |
| 00CB    | MOV R3, #0  | MOV R3, #0 | INITIALIZE REGISTER R3 |
| 00CC    | MOV R4, #0  | MOV R4, #0 | INITIALIZE REGISTER R4 |
| 00CD    | MOV R5, #0  | MOV R5, #0 | INITIALIZE REGISTER R5 |
| 00CE    | MOV R6, #0  | MOV R6, #0 | INITIALIZE REGISTER R6 |
| 00CF    | MOV R7, #0  | MOV R7, #0 | INITIALIZE REGISTER R7 |
| 00D0    | MOV A, #0   | MOV A, #0  | INITIALIZE ACCUMULATOR |
| 00D1    | MOV R0, #0  | MOV R0, #0 | INITIALIZE REGISTER R0 |
| 00D2    | MOV R1, #0  | MOV R1, #0 | INITIALIZE REGISTER R1 |
| 00D3    | MOV R2, #0  | MOV R2, #0 | INITIALIZE REGISTER R2 |
| 00D4    | MOV R3, #0  | MOV R3, #0 | INITIALIZE REGISTER R3 |
| 00D5    | MOV R4, #0  | MOV R4, #0 | INITIALIZE REGISTER R4 |
| 00D6    | MOV R5, #0  | MOV R5, #0 | INITIALIZE REGISTER R5 |
| 00D7    | MOV R6, #0  | MOV R6, #0 | INITIALIZE REGISTER R6 |
| 00D8    | MOV R7, #0  | MOV R7, #0 | INITIALIZE REGISTER R7 |
| 00D9    | MOV A, #0   | MOV A, #0  | INITIALIZE ACCUMULATOR |
| 00DA    | MOV R0, #0  | MOV R0, #0 | INITIALIZE REGISTER R0 |
| 00DB    | MOV R1, #0  | MOV R1, #0 | INITIALIZE REGISTER R1 |
| 00DC    | MOV R2, #0  | MOV R2, #0 | INITIALIZE REGISTER R2 |
| 00DD    | MOV R3, #0  | MOV R3, #0 | INITIALIZE REGISTER R3 |
| 00DE    | MOV R4, #0  | MOV R4, #0 | INITIALIZE REGISTER R4 |
| 00DF    | MOV R5, #0  | MOV R5, #0 | INITIALIZE REGISTER R5 |
| 00E0    | MOV R6, #0  | MOV R6, #0 | INITIALIZE REGISTER R6 |
| 00E1    | MOV R7, #0  | MOV R7, #0 | INITIALIZE REGISTER R7 |
| 00E2    | MOV A, #0   | MOV A, #0  | INITIALIZE ACCUMULATOR |
| 00E3    | MOV R0, #0  | MOV R0, #0 | INITIALIZE REGISTER R0 |
| 00E4    | MOV R1, #0  | MOV R1, #0 | INITIALIZE REGISTER R1 |
| 00E5    | MOV R2, #0  | MOV R2, #0 | INITIALIZE REGISTER R2 |
| 00E6    | MOV R3, #0  | MOV R3, #0 | INITIALIZE REGISTER R3 |
| 00E7    | MOV R4, #0  | MOV R4, #0 | INITIALIZE REGISTER R4 |
| 00E8    | MOV R5, #0  | MOV R5, #0 | INITIALIZE REGISTER R5 |
| 00E9    | MOV R6, #0  | MOV R6, #0 | INITIALIZE REGISTER R6 |
| 00EA    | MOV R7, #0  | MOV R7, #0 | INITIALIZE REGISTER R7 |
| 00EB    | MOV A, #0   | MOV A, #0  | INITIALIZE ACCUMULATOR |
| 00EC    | MOV R0, #0  | MOV R0, #0 | INITIALIZE REGISTER R0 |
| 00ED    | MOV R1, #0  | MOV R1, #0 | INITIALIZE REGISTER R1 |
| 00EE    | MOV R2, #0  | MOV R2, #0 | INITIALIZE REGISTER R2 |
| 00EF    | MOV R3, #0  | MOV R3, #0 | INITIALIZE REGISTER R3 |
| 00F0    | MOV R4, #0  | MOV R4, #0 | INITIALIZE REGISTER R4 |
| 00F1    | MOV R5, #0  | MOV R5, #0 | INITIALIZE REGISTER R5 |
| 00F2    | MOV R6, #0  | MOV R6, #0 | INITIALIZE REGISTER R6 |
| 00F3    | MOV R7, #0  | MOV R7, #0 | INITIALIZE REGISTER R7 |
| 00F4    | MOV A, #0   | MOV A, #0  | INITIALIZE ACCUM       |



# Appendix A. Automobile Turn-indicator Controller Program Listing.

ISIS-II MCS-51 MACRO ASSEMBLER V1.0  
OBJECT MODULE PLACED IN :F0:AP70.HEX  
ASSEMBLER INVOKED BY: :f1:asm51 ap70 src date(328)

| LOC  | OBJ | LINE | SOURCE   |
|------|-----|------|--|
|      |     | 1    | \$XREF TITLE(AP-70 APPENDIX)                               |
|      |     | 2    | ; *****  |
|      |     | 3    | ;  |
|      |     | 4    | ;  |
|      |     | 5    | THE FOLLOWING PROGRAM USES THE BOOLEAN INSTRUCTION SET     |
|      |     | 6    | OF THE INTEL 8051 MICROCOMPUTER TO PERFORM A NUMBER OF     |
|      |     | 7    | AUTOMOTIVE DASHBOARD CONTROL FUNCTIONS RELATING TO         |
|      |     | 8    | TURN SIGNAL CONTROL, EMERGENCY BLINKERS, BRAKE LIGHT       |
|      |     | 9    | CONTROL, AND PARKING LIGHT OPERATION.                      |
|      |     | 10   | THE ALGORITHMS AND HARDWARE ARE DESCRIBED IN DESIGN        |
|      |     | 11   | EXAMPLE #4 OF INTEL APPLICATION NOTE AP-70.                |
|      |     | 12   | "USING THE INTEL MCS-51(TM)                                |
|      |     | 13   | BOOLEAN PROCESSING CAPABILITIES"                           |
|      |     | 14   | ; *****  |
|      |     | 15   | ;  |
|      |     | 16   | INPUT PIN DECLARATIONS:                                    |
|      |     | 17   | (ALL INPUTS ARE POSITIVE-TRUE LOGIC.                       |
|      |     | 18   | INPUTS ARE HIGH WHEN RESPECTIVE SWITCH CONTACT IS CLOSED.) |
|      |     | 19   | ;  |
| 0090 |     | 20   | BRAKE BIT P1.0 ; BRAKE PEDAL DEPRESSED                     |
| 0091 |     | 21   | EMERG BIT P1.1 ; EMERGENCY BLINKER ACTIVATED               |
| 0092 |     | 22   | PARK BIT P1.2 ; PARKING LIGHTS ON                          |
| 0093 |     | 23   | L_TURN BIT P1.3 ; TURN LEVER DOWN                          |
| 0094 |     | 24   | R_TURN BIT P1.4 ; TURN LEVER UP                            |
|      |     | 25   | ;  |
|      |     | 26   | OUTPUT PIN DECLARATIONS:                                   |
|      |     | 27   | (ALL OUTPUTS ARE POSITIVE TRUE LOGIC.                      |
|      |     | 28   | BULB IS TURNED ON WHEN OUTPUT PIN IS HIGH.)                |
|      |     | 29   | ;  |
| 0095 |     | 30   | L_FRNT BIT P1.5 ; FRONT LEFT-TURN INDICATOR                |
| 0096 |     | 31   | R_FRNT BIT P1.6 ; FRONT RIGHT-TURN INDICATOR               |
| 0097 |     | 32   | L_DASH BIT P1.7 ; DASHBOARD LEFT-TURN INDICATOR            |
| 00A0 |     | 33   | R_DASH BIT P2.0 ; DASHBOARD RIGHT-TURN INDICATOR           |
| 00A1 |     | 34   | L_REAR BIT P2.1 ; REAR LEFT-TURN INDICATOR                 |
| 00A2 |     | 35   | R_REAR BIT P2.2 ; REAR RIGHT-TURN INDICATOR                |
|      |     | 36   | ;  |
| 00A3 |     | 37   | S_FAIL BIT P2.3 ; ELECTRICAL SYSTEM FAULT INDICATOR        |
|      |     | 38   | ;  |
|      |     | 39   | INTERNAL VARIABLE DEFINITIONS:                             |
|      |     | 40   | ;  |
| 0020 |     | 41   | SUB_DIV DATA 20H ; INTERRUPT RATE SUBDIVIDER               |
| 0000 |     | 42   | HI_FREQ BIT SUB_DIV.0 ; HIGH-FREQUENCY OSCILLATOR BIT      |
| 0007 |     | 43   | LO_FREQ BIT SUB_DIV.7 ; LOW-FREQUENCY OSCILLATOR BIT       |
|      |     | 44   | ;  |
| 00D1 |     | 45   | DIM BIT PSW.1 ; PARKING LIGHTS ON FLAG                     |
|      |     | 46   | ;  |
|      |     | 47   | ; =====  |
|      |     | 48   | +1 \$EJECT   |

| LOC  | OBJ    | LINE | SOURCE  |
|------|--------|------|---|
| 0000 | 020040 | 49   | ORG 0000H ; RESET VECTOR  |
|      |        | 50   | LJMP INIT   |
| 000B |        | 51   |   |
| 000B | 758CF0 | 52   | ORG 000BH ; TIMER 0 SERVICE VECTOR                                      |
| 000E | C0D0   | 53   | MOV TH0, #-16 ; HIGH TIMER BYTE ADJUSTED TO CONTROL INT. RATE           |
| 0010 | 0154   | 54   | PUSH PSW ; EXECUTE CODE TO SAVE ANY REGISTERS USED BELOW                |
|      |        | 55   | AJMP UPDATE ; (CONTINUE WITH REST OF ROUTINE)                           |
| 0040 |        | 56   |   |
| 0040 | 758A00 | 57   | ORG 0040H   |
| 0043 | 758CF0 | 58   | INIT: MOV TLO, #0 ; ZERO LOADED INTO LOW-ORDER BYTE AND                 |
| 0046 | 758961 | 59   | MOV TH0, #-16 ; -16 IN HIGH-ORDER BYTE GIVES 4 MSEC PERIOD              |
|      |        | 60   | MOV TMOD, #01100001B ; 8-BIT AUTO RELOAD COUNTER MODE FOR TIMER 1,      |
|      |        | 61   | 16-BIT TIMER MODE FOR TIMER 0 SELECTED                                  |
| 0049 | 7520F4 | 62   | MOV SUB_DIV, #244 ; SUBDIVIDE INTERRUPT RATE BY 244 FOR 1 HZ            |
| 004C | D2A9   | 63   | SETB ETO ; USE TIMER 0 OVERFLOWS TO INTERRUPT PROGRAM                   |
| 004E | D2AF   | 64   | SETB EA ; CONFIGURE IE TO GLOBALLY ENABLE INTERRUPTS                    |
| 0050 | D2BC   | 65   | SETB TRO ; KEEP INSTRUCTION CYCLE COUNT UNTIL OVERFLOW                  |
| 0052 | 80FE   | 66   | SJMP \$ ; START BACKGROUND PROGRAM EXECUTION                            |
|      |        | 67   |   |
|      |        | 68   |   |
| 0054 | D52038 | 69   | UPDATE: DJNZ SUB_DIV, TOSERV ; EXECUTE SYSTEM TEST ONLY ONCE PER SECOND |
| 0057 | 7520F4 | 70   | MOV SUB_DIV, #244 ; GET VALUE FOR NEXT ONE SECOND DELAY AND             |
|      |        | 71   | GO THROUGH ELECTRICAL SYSTEM TEST CODE:                                 |
| 005A | 4390E0 | 72   | ORL P1, #11100000B ; SET CONTROL OUTPUTS HIGH                           |
| 005D | 43A007 | 73   | ORL P2, #00000111B  |
| 0060 | C295   | 74   | CLR L_FRNT ; FLOAT DRIVE COLLECTOR                                      |
| 0062 | 20B428 | 75   | JB TO, FAULT ; TO SHOULD BE PULLED LOW                                  |
| 0065 | D295   | 76   | SETB L_FRNT ; PULL COLLECTOR BACK DOWN                                  |
| 0067 | C297   | 77   | CLR L_DASH ; REPEAT SEQUENCE FOR L_DASH                                 |
| 0069 | 20B421 | 78   | JB TO, FAULT  |
| 006C | D297   | 79   | SETB L_DASH   |
| 006E | C2A1   | 80   | CLR L_REAR ; L_REAR,  |
| 0070 | 20B41A | 81   | JB TO, FAULT  |
| 0073 | D2A1   | 82   | SETB L_REAR   |
| 0075 | C296   | 83   | CLR R_FRNT ; R_FRNT,  |
| 0077 | 20B413 | 84   | JB TO, FAULT  |
| 007A | D296   | 85   | SETB R_FRNT   |
| 007C | C2A0   | 86   | CLR R_DASH ; R_DASH,  |
| 007E | 20B40C | 87   | JB TO, FAULT  |
| 0081 | D2A0   | 88   | SETB R_DASH   |
| 0083 | C2A2   | 89   | CLR R_REAR ; AND R_REAR.  |
| 0085 | 20B405 | 90   | JB TO, FAULT  |
| 0088 | D2A2   | 91   | SETB R_REAR   |
|      |        | 92   |   |
|      |        | 93   | WITH ALL COLLECTORS GROUNDED, TO SHOULD BE HIGH                         |
|      |        | 94   | IF SO, CONTINUE WITH INTERRUPT ROUTINE.                                 |
|      |        | 95   |   |
| 008A | 20B402 | 96   | JB TO, TOSERV   |
| 008D | B2A3   | 97   | FAULT: CPL S_FAIL ; ELECTRICAL FAILURE PROCESSING ROUTINE               |
|      |        | 98   | (TOGGLE INDICATOR ONCE PER SECOND)                                      |
|      |        | 99   | +1 \$EJECT  |

| LOC         | OBJ | LINE | SOURCE  |
|-------------|-----|------|---|
| 008D BSV3   |     | 100  | CONTINUE WITH INTERRUPT PROCESSING.                       |
| 008V 505+05 |     | 101  |   |
|             |     | 102  | 1) COMPUTE LOW BULB INTENSITY WHEN PARKING LIGHTS ARE ON. |
|             |     | 103  |   |
| 008F A201   |     | 104  | TOSERV: MOV C, SUB_DIV. 1 ; START WITH 50 PERCENT,        |
| 0091 8200   |     | 105  | ANL C, SUB_DIV. 0 ; MASK DOWN TO 25 PERCENT,              |
| 0093 7202   |     | 106  | ORL C, SUB_DIV. 2 ; BUILD BACK TO 62.5 PERCENT,           |
| 0095 8292   |     | 107  | ANL C, PARK ; GATE WITH PARKING LIGHT SWITCH,             |
| 0097 92D1   |     | 108  | MOV DIM, C ; AND SAVE IN TEMP. VARIABLE.                  |
|             |     | 109  |   |
| 009E 508+0C |     | 110  | 2) COMPUTE AND OUTPUT LEFT-HAND DASHBOARD INDICATOR.      |
|             |     | 111  |   |
| 0099 A293   |     | 112  | MOV C, L_TURN ; SET CARRY IF TURN                         |
| 009B 7291   |     | 113  | ORL C, EMERG ; OR EMERGENCY SELECTED.                     |
| 009D 8207   |     | 114  | ANL C, L_FREQ ; IF SO, GATE IN 1 HZ SIGNAL                |
| 009F 9297   |     | 115  | MOV L_DASH, C ; AND OUTPUT TO DASHBOARD.                  |
|             |     | 116  |   |
| 009E 508+1V |     | 117  | 3) COMPUTE AND OUTPUT LEFT-HAND FRONT TURN SIGNAL.        |
|             |     | 118  |   |
| 009A 92D5   |     | 119  | MOV FO, C ; SAVE FUNCTION SO FAR.                         |
| 00A3 72D1   |     | 120  | ORL C, DIM ; ADD IN PARKING LIGHT FUNCTION                |
| 00A5 9295   |     | 121  | MOV L_FRNT, C ; AND OUTPUT TO TURN SIGNAL.                |
|             |     | 122  |   |
| 0095 508+5B |     | 123  | 4) COMPUTE AND OUTPUT LEFT-HAND REAR TURN SIGNAL.         |
|             |     | 124  |   |
| 00A7 A290   |     | 125  | MOV C, BRAKE ; GATE BRAKE PEDAL SWITCH                    |
| 00A9 B093   |     | 126  | ANL C, /L_TURN ; WITH TURN LEVER.                         |
| 00AB 72D5   |     | 127  | ORL C, FO ; INCLUDE TEMP. VARIABLE FROM DASH              |
| 00AD 72D1   |     | 128  | ORL C, DIM ; AND PARKING LIGHT FUNCTION                   |
| 00AF 92A1   |     | 129  | MOV L_REAR, C ; AND OUTPUT TO TURN SIGNAL.                |
|             |     | 130  |   |
| 0095 508+5B |     | 131  | 5) REPEAT ALL OF ABOVE FOR RIGHT-HAND COUNTERPARTS.       |
|             |     | 132  |   |
| 00B1 A294   |     | 133  | MOV C, R_TURN ; SET CARRY IF TURN                         |
| 00B3 7291   |     | 134  | ORL C, EMERG ; OR EMERGENCY SELECTED.                     |
| 00B5 8207   |     | 135  | ANL C, L_FREQ ; IF SO, GATE IN 1 HZ SIGNAL                |
| 00B7 92A0   |     | 136  | MOV R_DASH, C ; AND OUTPUT TO DASHBOARD.                  |
| 00B9 92D5   |     | 137  | MOV FO, C ; SAVE FUNCTION SO FAR.                         |
| 00BB 72D1   |     | 138  | ORL C, DIM ; ADD IN PARKING LIGHT FUNCTION                |
| 00BD 9296   |     | 139  | MOV R_FRNT, C ; AND OUTPUT TO TURN SIGNAL.                |
| 00BF A290   |     | 140  | MOV C, BRAKE ; GATE BRAKE PEDAL SWITCH                    |
| 00C1 B094   |     | 141  | ANL C, /R_TURN ; WITH TURN LEVER.                         |
| 00C3 72D5   |     | 142  | ORL C, FO ; INCLUDE TEMP. VARIABLE FROM DASH              |
| 00C5 72D1   |     | 143  | ORL C, DIM ; AND PARKING LIGHT FUNCTION                   |
| 00C7 92A2   |     | 144  | MOV R_REAR, C ; AND OUTPUT TO TURN SIGNAL.                |
|             |     | 145  |   |
|             |     | 146  | RESTORE STATUS REGISTER AND RETURN.                       |
|             |     | 147  |   |
| 00C9 D0D0   |     | 148  | POP PSW ; RESTORE PSW                                     |
| 00CB 32     |     | 149  | RETI ; AND RETURN FROM INTERRUPT ROUTINE                  |
|             |     | 150  |   |
|             |     | 151  | END   |

# XREF SYMBOL TABLE LISTING

| NAME    | TYPE   | VALUE AND REFERENCES                 |
|---------|--------|--------------------------------------|
| BRAKE . | N BSEG | 0090H 20# 125 140                    |
| DIM .   | N BSEG | 00D1H 45# 108 120 128 138 143        |
| EA .    | N BSEG | 00AFH 64                             |
| EMERG . | N BSEG | 0091H 21# 113 134                    |
| ETO .   | N BSEG | 00A9H 63                             |
| FO .    | N BSEG | 00D5H 119 127 137 142                |
| FAULT . | L CSEG | 00BDH 75 78 81 84 87 90 97#          |
| HI_FREQ | N BSEG | 0000H 42#                            |
| INIT .  | L CSEG | 0040H 50 58#                         |
| L_DASH. | N BSEG | 0097H 32# 77 79 115                  |
| L_FRNT. | N BSEG | 0095H 30# 74 76 121                  |
| L_REAR. | N BSEG | 00A1H 34# 80 82 129                  |
| L_TURN. | N BSEG | 0093H 23# 112 126                    |
| LO_FREQ | N BSEG | 0007H 43# 114 135                    |
| P1 .    | N DSEG | 0090H 20 21 22 23 24 30 31 32 72     |
| P2 .    | N DSEG | 00A0H 33 34 35 37 73                 |
| PARK .  | N BSEG | 0092H 22# 107                        |
| PSW .   | N DSEG | 00D0H 45 54 148                      |
| R_DASH. | N BSEG | 00A0H 33# 86 88 136                  |
| R_FRNT. | N BSEG | 0096H 31# 83 85 139                  |
| R_REAR. | N BSEG | 00A2H 35# 89 91 144                  |
| R_TURN. | N BSEG | 0094H 24# 133 141                    |
| S_FAIL. | N BSEG | 00A3H 37# 97                         |
| SUB_DIV | N DSEG | 0020H 41# 42 43 62 69 70 104 105 106 |
| TO .    | N BSEG | 00B4H 75 78 81 84 87 90 96           |
| TOSERV. | L CSEG | 00BFH 69 96 104#                     |
| THO .   | N DSEG | 00BCH 53 59                          |
| TLO .   | N DSEG | 00BAH 58                             |
| TMOD.   | N DSEG | 00B9H 60                             |
| TRO .   | N BSEG | 00BCH 65                             |
| UPDATE. | L CSEG | 0054H 55 69#                         |

ASSEMBLY COMPLETE. NO ERRORS FOUND



October 1985

VERBODEN TOEGANG: NO ENGLISH

DESIGNING WITH THE  
80C51BH

by Tom Williamson  
MCO Applications Engineer



## CMOS Evolves

The original CMOS logic families were the 4000-series and the 74C-series circuits. The 74C-series circuits are functional equivalents to the correspondingly numbered 74-series TTL circuits, but have CMOS logic levels and retain the other well known characteristics of CMOS logic.

These characteristics are: low power consumption, high noise immunity, and slow speed. The low power consumption is inherent to the nature of the CMOS circuit. The noise immunity is due partly to the CMOS logic levels, and partly to the slowness of the circuits. The slow speed is due to the technology used to construct the transistors in the circuit.

The technology used is called metal-gate CMOS, because the transistor gates are formed by metal deposition. More importantly, the gates are formed after the drain and source regions have been defined, and must overlap the source and drain somewhat to allow for alignment tolerances. This overlap plus the relatively large size of the transistors themselves result in high electrode capacitance, and that is what limits the speed of the circuit.

High speed CMOS became feasible with the development of the self-aligning silicon gate technology. In this process polysilicon gates are deposited **before** the source and drain regions are defined. Then the source and drain regions are formed by ion implantation using the gate itself as a mask for the implantation. This eliminates most of the overlap capacitance. In addition, the process allows smaller transistors. The result is a significant increase in circuit speed. The 74HC-series of CMOS logic circuits is based on this technology, and has speeds comparable to LS TTL, which is to say about 10 times faster than the 74C-series circuits.

The size reduction that contributes to the higher speed also demands an accompanying reduction in the maximum supply voltage. High-speed CMOS is generally limited to 6V.

## What Is CHMOS?

CHMOS is the name given to Intel's high-speed CMOS processes. There are two CHMOS processes, one based on an n-well structure and one based on a p-well structure. In the n-well structure, n-type wells are diffused into a p-type substrate. Then the n-channel transistors (nFETs) are built into the substrate and pFETs are built into the n-wells. In the p-well structure, p-type wells are diffused into an n-type substrate. Then the nFETs are built into the wells and

pFETs, into the substrate. Both processes have their advantages and disadvantages, which are largely transparent to the user.

Lower operating voltages are easier to obtain with the p-well structure than with the n-well structure. But the p-well structure does not easily adapt to an EPROM which would be pin-for-pin compatible with HMOS EPROMs. On the other hand the n-well structure can be based on the solidly founded HMOS process, in which nFETs are built into a p-type substrate. This allows somewhat more than half of the transistors in a CHMOS chip to be constructed by processes that are already well characterized.

Currently Intel's CHMOS microcontrollers and memory products are n-well devices, whereas CHMOS microprocessors are p-well devices.

Further discussion of the CHMOS technology is provided in references 1 and 2 (which are reprinted in the Microcontroller Handbook).

## The MCS®-51 Family in CHMOS

The 80C51BH is the CHMOS version of Intel's original 8051. The 80C31BH is the ROMless 80C51BH, equivalent to the 8031. These CHMOS devices are architecturally identical with their HMOS counterparts, except that they have two added features for reduced power. These are the Idle and Power Down modes of operation.

In most cases, an 80C51BH can directly replace the 8051 in existing applications. It can execute the same code at the same speed, accept signals from the same sources, and drive the same loads. However, the 80C51BH covers a wider range of speeds, will emit CMOS logic levels to CMOS loads, and will draw about 1/10 the current of an 8051 (and less yet in the reduced power modes). Interchangeability between the HMOS and CHMOS devices is discussed in more detail in the final section of this Application Note.

It should be noted that the 80C51BH CPU is not static. That means if the clock frequency is too low, the CPU might forget what it was doing. This is because the circuitry uses a number of dynamic nodes. A dynamic node is one that uses the node-to-ground capacitance to form a temporary storage cell. Dynamic nodes are used to reduce the transistor count, and hence the chip area, thus to produce a more economical device.

This is not to say that the on-chip RAM in CHMOS microcontrollers is dynamic. It's not. It's the CPU that is dynamic, and that is what imposes the minimum clock frequency specification.

## Latchup

Latchup is an SCR-type turn-on phenomenon that is the traditional nemesis of CMOS systems. The substrate, the wells, and the transistors form parasitic pnpn structures within the device. These parasitic structures turn on like an SCR if a sufficient amount of forward current is driven through one of the junctions. From the circuit designer's point of view it can happen whenever an input or output pin is externally driven a diode drop above  $V_{CC}$  or below  $V_{SS}$ , by a source that is capable of supplying the required trigger current.

However much of a problem latchup has been in the past, it is good to know that in most recently developed CMOS devices, and specifically in CHMOS devices, the current required to trigger latchup is typically well over 100 mA. The 80C51BH is virtually immune to latchup. (References 1 and 2 present a discussion of the latchup mechanisms and the steps that are taken on the chip to guard against it.) Modern CMOS is not absolutely immune to latchup, but with trigger currents in the hundreds of mA, latchup is certainly a lot easier to avoid than it once was.

A careless power-up sequence might trigger a latchup in the older CMOS families, but it's unlikely to be a major problem in high-speed CMOS or in CHMOS. There is still some risk incurred in inserting or removing chips or boards in a CMOS system while the power is on. Also, severe transients, such as inductive kicks or momentary short-circuits, can exceed the trigger current for latchup.

For applications in which some latchup risk seems unavoidable, you can put a small resistor (100 ohms or so) in series with signal lines to ensure that the trigger current will never be reached. This also helps to control overshoot and RFI.

## Logic Levels and Interfacing Problems

CMOS logic levels differ from TTL levels in two ways.

First, for equal supply voltages, CMOS gives (and requires) a higher "logic 1" level than TTL. Secondly, CMOS logic levels are  $V_{CC}$  (or  $V_{DD}$ ) dependent, whereas guaranteed TTL logic levels are fixed when  $V_{CC}$  is within TTL specs.

Standard 74HC logic levels are as follows:

$$\begin{aligned} V_{IH} \text{ MIN} &= 70\% \text{ of } V_{CC} \\ V_{IL} \text{ MAX} &= 20\% \text{ of } V_{CC} \\ V_{OH} \text{ MIN} &= V_{CC} - 0.1V, |I_{OH}| \leq 20 \mu A \\ V_{OL} \text{ MAX} &= 0.1V, |I_{OL}| \leq 20 \mu A \end{aligned}$$

Figure 1 compares 74HC, LS TTL, and 74HCT logic levels with those of the HMOS 8051 and the CHMOS 80C51BH for  $V_{CC} = 5V$ .

Output logic levels depend of course on load current, and are normally specified at several load currents. When CMOS and TTL are powered by the same  $V_{CC}$ , the logic levels guaranteed on the data sheets indicate that CMOS can drive TTL, but TTL can't drive CMOS. The incompatibility is that the TTL circuit's  $V_{OH}$  level is too low to reliably be recognized by the CMOS circuit as a valid  $V_{IH}$ .

Since HMOS circuits were designed to be TTL-compatible, they have the same incompatibility.

Fortunately, 74HCT-series circuits are available to ease these interfacing problems. They have TTL-compatible logic levels at the inputs and standard CMOS levels at the outputs.

The 80C51BH is designed to work with either TTL or CMOS. Therefore its logic levels are specified very much like 74HCT circuits. That is, its input logic levels are TTL-compatible, and its output characteristics are like standard high-speed CMOS.

## Noise Considerations

One of the major reasons for going to CMOS has traditionally been that CMOS is less susceptible to noise. As previously noted, its low susceptibility to

| Logic State: | $V_{CC} = 5V$ |       |        |       |         |
|--------------|---------------|-------|--------|-------|---------|
|              | 74HC          | 74HCT | LS TTL | 8051  | 80C51BH |
| $V_{IH}$     | 3.5V          | 2.0V  | 2.0V   | 2.0V  | 1.9V    |
| $V_{IL}$     | 1.0V          | 0.8V  | 0.8V   | 0.8V  | 0.9V    |
| $V_{OH}$     | 4.9V          | 4.9V  | 2.7V   | 2.4V  | 4.5V    |
| $V_{OL}$     | 0.1V          | 0.1V  | 0.5V   | 0.45V | 0.45V   |

**Figure 1. Logic Level Comparison.** (Output voltage levels depend on load current. Data sheets list guaranteed output levels for several load currents. The output levels listed here are for minimum loading.)



noise is partly due to superior noise margins, and partly due to its slow speed.

Noise margin is the difference between  $V_{OL}$  and  $V_{IL}$ , or between  $V_{OH}$  and  $V_{IH}$ . If  $V_{OH}$  from a driving circuit is 2.7V and  $V_{IH}$  to the driven circuit is 2.0V, then the driven circuit has 0.7V of noise margin at the logic high level. These kinds of comparisons show that an all-CMOS system has wider noise margins than an all-TTL system.

Figure 2 shows noise margins in CMOS and LS TTL systems when both have  $V_{CC} = 5V$ . It can be seen that CMOS/CMOS and CMOS/CHMOS systems have an edge over LS TTL in this respect.

Noise margins can be misleading, however, because they don't say how much noise energy it takes to induce in the circuit a noise voltage of sufficient amplitude to cause a logic error. This would involve consideration of the width of the noise pulse as compared with the circuit's response speed, and the impedance to ground from the point of noise introduction in the circuit.

When these considerations are included, it is seen that using the slower 74C- and 4000-series circuits with a 12 or 15 volt supply voltage does offer a truly improved level of noise immunity, but that high-speed CMOS at 5V is not significantly better than TTL.

One should not mistake the wider supply voltage tolerance of high-speed CMOS for  $V_{CC}$  glitch immunity. Supply voltage tolerance is a DC rating, not a glitch rating.

For any clocked CMOS, and most especially for VLSI CMOS,  $V_{CC}$  decoupling is critical. CHMOS draws

current in extremely sharp spikes at the clock edges. The VHF and UHF components of these spikes are not drawn from the power supply, but from the decoupling capacitor. If the decoupling circuit is not sufficiently low in inductance,  $V_{CC}$  will glitch at each clock edge. We suggest that a 0.1  $\mu F$  decoupler cap be used in a minimum-inductance configuration with the microcontroller. A minimum-inductance configuration is one that minimizes the area of the loop formed by the chip ( $V_{CC}$  to  $V_{SS}$ ), the traces to the decoupler cap, and the decoupler cap. PCB designers too often fail to understand that if the traces that connect the decoupler cap to the  $V_{CC}$  and  $V_{SS}$  pins aren't short and direct, the decoupler loses much of its effectiveness.

Overshoot and ringing in signal lines are potential sources of logic upsets. These can largely be controlled by circuit layout. Inserting small resistors (about 100 ohms) in series with signal lines that seem to need them will also help.

The sharp edges produced by high-speed CMOS can cause RFI problems. The severity of these problems is largely a function of the PCB layout. We don't mean to imply that all RFI problems can be solved by a better PCB layout. It may well be, for example, that in some RFI-sensitive designs high-speed CMOS is simply not the answer. But circuit layout is a critical factor in the noise performance of any electronic system, and more so in high-speed CMOS systems than others.

Circuit layout techniques for minimizing noise susceptibility and generation are discussed in references 3 through 6.

### Unused Pins

CMOS input pins should not be left to float, but should always be pulled to one logic level or the other. If they float, they tend to float into the transition region between 0 and 1, where the pullup and pulldown devices in the input buffer are both conductive. This causes a significant increase in  $I_{CC}$ . A similar effect exists in HMOS circuits, but with less noticeable results.

In 80C51BH and 80C31BH designs, unused pins of Ports 1, 2, and 3 can be ignored, because they have internal pullups that will hold them at a valid Logic 1 level. Port 0 pins are different, however, in not having internal pullups (except during bus operations).

When the 80C51BH is in reset, the Port 0 pins are in a float state unless they are externally pulled up or down. If it's going to be held in reset for just a short time, the transient float state can probably be ignored. When it comes out of reset, the pins stay afloat unless

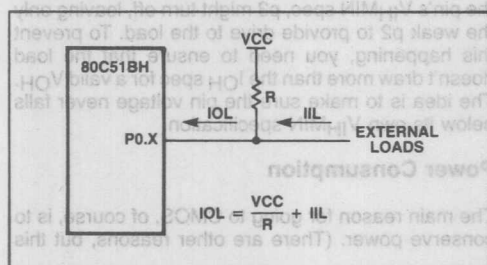
| Interface        | Noise Margin for<br>$V_{CC} = 5V$ |                               |
|------------------|-----------------------------------|-------------------------------|
|                  | Logic Low<br>$V_{IL}-V_{OL}$      | Logic High<br>$V_{OH}-V_{IH}$ |
| 74HC to 74HC     | 0.9V                              | 1.4V                          |
| LSTTL to LSTTL   | 0.3V                              | 0.7V                          |
| LSTTL to 74HCT   | 0.3V                              | 0.7V                          |
| LSTTL to 80C51BH | 0.3V                              | 0.7V                          |
| 74HC to 80C51BH  | 0.8V                              | 3.0V                          |
| 80C51BH to 74HC  | 0.8V                              | 1.0V                          |

Figure 2. Noise margins for CMOS and LS TTL circuits.

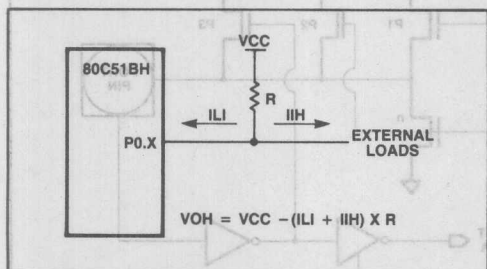
they are externally pulled either up or down. Alternatively, the software can internally write 0s to whatever Port 0 pins may be unused.

The same considerations are applicable to the 80C31BH with regards to reset. But when the 80C31BH comes out of reset, it commences bus operations, during which the logic levels at the pins are always well defined as high or low.

Consider the 80C31BH in the Power Down or Idle modes, however. In those modes it is not fetching instructions, and the Port 0 pins will float if not externally pulled high or low. The choice of whether to pull them high or low is the designer's. Normally it is sufficient to pull them up to  $V_{CC}$  with 10k resistors. But if power is going to be removed from circuits that are connected to the bus, it will be advisable to pull the bus pins down (normally with 10k resistors). Considerations involved in selecting pullup and pulldown resistor values are as follows.



**Figure 3a. Conditions defining the minimum value for R. P0.X is emitting a logic low. R must be large enough to not cause IOL to exceed data sheet specifications.**



**Figure 3b. Conditions defining the maximum value for R. P0.X is in a high impedance state. R must be small enough to keep VOH acceptably high.**

## Pullup Resistors

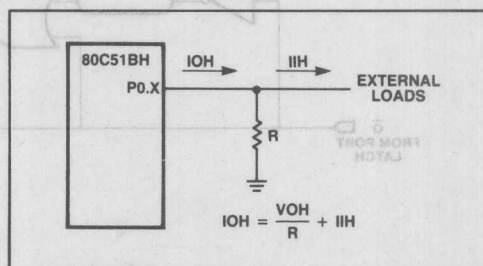
If a pullup resistor is to be used on a Port 0 pin, its minimum value is determined by  $IOL$  requirements. If the pin is trying to emit a 0, then it will have to sink the current from the pullup resistor plus whatever other current may be sourced by other loads connected to the pin, as shown in Figure 3A, while maintaining a valid output low ( $VOL$ ). To guarantee that the pin voltage will not exceed 0.45V, the resistor should be selected so that  $IOL$  doesn't exceed the value specified on the data sheet. In most CMOS applications, the minimum value would be about 2k ohms.

The maximum value you could use depends on how fast you want the pin to pull up after bus operations have ceased, and how high you want the  $VOH$  level to be. The smaller the resistor the faster it pulls up. Its effect on the  $VOH$  level is that  $VOH = VCC - (ILI + IIH) \times R$ .  $ILI$  is the input leakage current to the Port 0 pin, and  $IIH$  is the input high current to the external loads, as shown in Figure 3B. Normally  $VOH$  can be expected to reach  $0.9V_{CC}$  if the pullup resistance does not exceed about 50k ohms.

## Pulldown Resistors

If a pulldown resistor is to be used on a Port 0 pin, its minimum value is determined by  $VOH$  requirements during bus operations, and its maximum value is in most cases determined by leakage current.

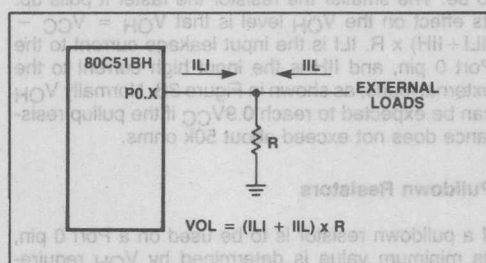
During bus operations the port uses internal pullups to emit 1s. The D.C. Characteristics in the data sheet list guaranteed  $VOH$  levels for given  $IOL$  currents. (The "-" sign in the  $IOL$  value means the pin is sourcing that current to the external load, as shown in Figure 4.) To ensure the  $VOH$  level listed in the data sheet, the resistor has to satisfy



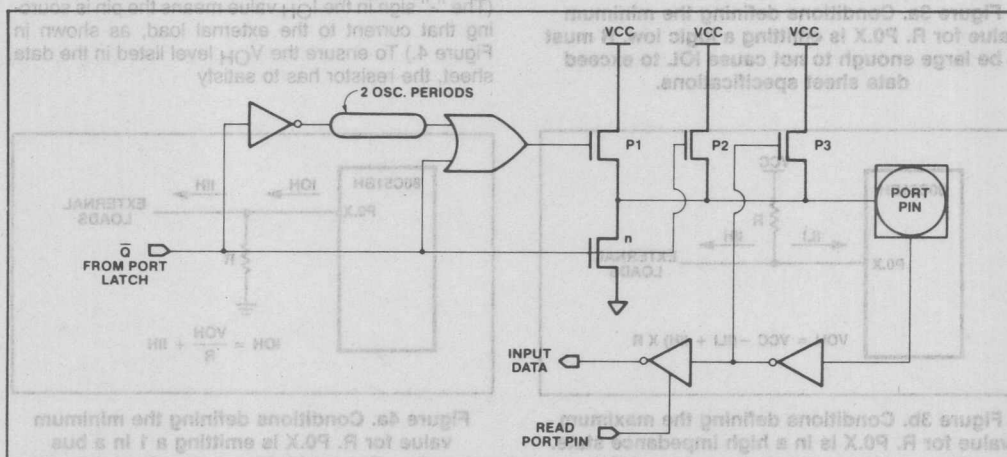
**Figure 4a. Conditions defining the minimum value for R. P0.X is emitting a 1 in a bus operation. R must be large enough to not cause IOH to exceed data sheet specifications.**

$$\frac{I_{OH}}{R} + I_{IH} \leq |I_{OL}|$$

where  $I_{IH}$  is the input high current to the external loads. If the pin is trying to emit a 0, then it will have to sink the current from the pullup resistor plus whatever the current from the pullup resistor is. When the pin goes into a high impedance state, the pulldown resistor will have to sink leakage current from the pin, plus whatever other current may be sourced by other loads connected to the pin, as shown in Figure 4B. The Port 0 leakage current is  $I_{LL}$  on the data sheet. The resistor should be selected so that the voltage developed across it by these currents will be seen as a logic low by whatever circuits are connected to it (including the 80C51BH). In CMOS/CHMOS applications, 50k ohms is normally a reasonable maximum value.



**Figure 4b. Conditions defining the maximum value for R. P0.X is in a high impedance state. R must be small enough to keep VOL acceptably low.**



**Figure 5. 80C51BH Output Drivers for Ports 1, 2 and 3.**

## Drive Capability of the Internal Pullups

There's an important difference between HMOS and CHMOS port drivers. The pins of Ports 1, 2, and 3 of the CHMOS parts each have three pullups: strong, normal, and weak, as shown in Figure 5. The strong pullup (p1) is only used during 0-to-1 transitions, to hasten the transition. The weak pullup (p2) is on whenever the bit latch contains a '1'. The "normal" pullup (p3) is controlled by the pin voltage itself.

The reason that p3 is controlled by the pin voltage is that if the pin is being used as an input, and the external source pulls it to a low, then turning off p3 makes for a lower  $I_{LL}$ . The data sheet shows an " $I_{TL}$ " specification. This is the current that p3 will source during the time the pin voltage is making its 1-to-0 transition. This is what  $I_{LL}$  would be if an input low at the pin didn't turn p3 off.

Note, however, that this p3 turn-off mechanism puts a restriction on the drive capacity of the pin if it's being used as an output. If you're trying to output a logic high, and the external load pulls the pin voltage below the pin's  $V_{IHMIN}$  spec, p3 might turn off, leaving only the weak p2 to provide drive to the load. To prevent this happening, you need to ensure that the load doesn't draw more than the  $I_{OH}$  spec for a valid  $V_{OH}$ . The idea is to make sure the pin voltage never falls below its own  $V_{IHMIN}$  specification.

## Power Consumption

The main reason for going to CMOS, of course, is to conserve power. (There are other reasons, but this

is the main one.) Conserving power doesn't mean just reducing your electric bill. Nor does it necessarily relate to battery operation, although battery operation without CMOS is pretty unhandy. The main reason for conserving power is to be able to put more functionality into a smaller space. The reduced power consumption allows the use of smaller and lighter power supplies, and less heat being generated allows denser packaging of circuit components. Expensive fans and blowers can usually be eliminated.

A cooler running chip is also more reliable, since most random and wearout failures relate to die temperature. And finally, the lower power dissipation will allow more functions to be integrated onto the chip.

The reason CMOS consumes less power than NMOS is that when it's in a stable state there is no path of conduction from  $V_{CC}$  to  $V_{SS}$  except through various leakage paths. CMOS does draw current when it's changing states. How much current it draws depends on how often and how quickly it changes states.

CMOS circuits draw current in sharp spikes during

logical transitions. These current spikes are made up of two components. One is the current that flows during the transition time when pullup and pulldown FETs are both active. The average (DC) value of this component is larger when the transition times of the input signals are longer. For this reason, if the current draw is a critical factor in the design, slow rise and fall times should be avoided, even when the system speed doesn't seem to justify a need for nanosecond switching speeds.

The other component is the current that charges stray and load capacitance at the nodes of a CMOS logic gate. The average value of this current spike is its area (integral over time) multiplied by its rep rate. Its area is the amount of charge it takes to raise the node capacitance,  $C$ , to  $V_{CC}$ . That amount of charge is just  $C \times V_{CC}$ . So the average value of the current spike is  $C \times V_{CC} \times f$ , where  $f$  is the clock frequency.

This component of current increases linearly with clock frequency. For minimal current draw, the 80C51BH-2 is spec'd to run at frequencies as low as 500kHz.

Keep in mind, though, that other component of current that is due to slow rise and fall times. A sinusoid is not the optimal waveform to drive the XTAL1 pin with. Yet crystal oscillators, including the one on the 80C51BH, generate sinusoidal waveforms. Therefore, if the on-chip oscillator is being used, you can expect the device to draw more current at 500kHz than it does at 1.5MHz, as shown in Figure 6. If you derive a good sharp square wave from an external oscillator, and use that to drive XTAL1, then the microcontroller will draw less current. But the external oscillator will probably make up the difference.

The 80C51BH has two power-saving features not available in the HMOS devices. These are the Idle and Power Down modes of operation. The on-chip hardware that implements these reduced power modes is shown in Figure 7. Both modes are invoked by software.

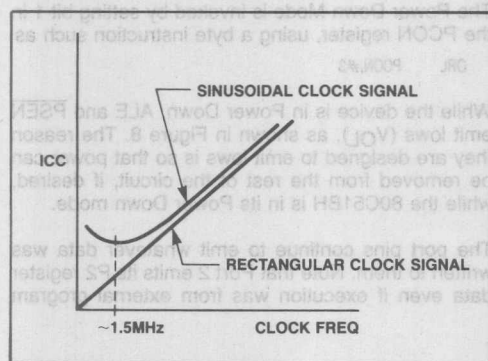


Figure 6. 80C51BH ICC vs. clock frequency.

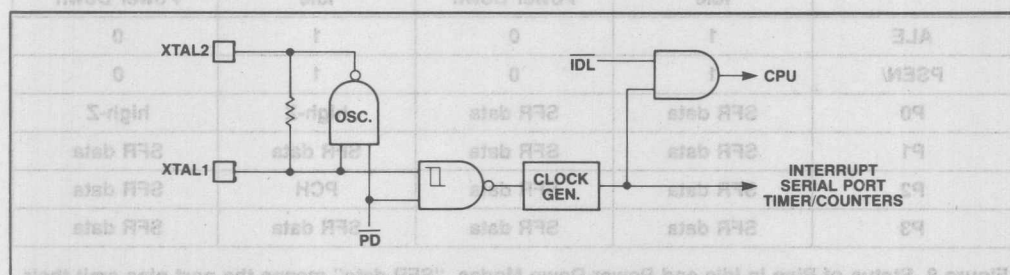


Figure 7. Oscillator and Clock Circuitry showing Idle and Power Down hardware.



**Idle:** In the Idle Mode ( $\overline{IDL} = 0$  in Figure 7), the CPU puts itself to sleep by gating off its own clock. It doesn't stop the oscillator. It just stops the internal clock signal from getting to the CPU. Since the CPU draws 80 to 90 percent of the chip's power, shutting it off represents a fairly significant power savings. The on-chip peripherals (timers, serial port, interrupts, etc.) and RAM continue to function as normal. The CPU status is preserved in its entirety: the Stack Pointer, Program Counter, Program Status Word, Accumulator, and all other registers maintain their data during Idle.

The Idle Mode is invoked by setting bit 0 (IDL) of the PCON register. PCON is not bit-addressable, so the bit has to be set by a byte operation, such as

```
ORL    PCON,#1
```

The PCON register also contains flag bits GF0 and GF1, which can be used for any general purposes, or to give an indication if an interrupt occurred during normal operation or during Idle. In this application, the instruction that invokes Idle also sets one or both of the flag bits. Their status can then be checked in the interrupt routines.

While the device is in the Idle mode, ALE and PSEN emit logic high ( $V_{OH}$ ), as shown in Figure 8. This is so external EPROM can be deselected and have its output disabled.

The port pins hold the logical states they had at the time the Idle was activated. If the device was executing out of external program memory, Port 0 is left in a high impedance state and Port 2 continues to emit the high byte of the program counter (using the strong pullups to emit 1s). If the device was executing out of internal program memory, Ports 0 and 2 continue to emit whatever is in the P0 and P2 registers.

| Pin   | Internal Execution |            | External Execution |            |
|-------|--------------------|------------|--------------------|------------|
|       | Idle               | Power Down | Idle               | Power Down |
| ALE   | 1                  | 0          | 1                  | 0          |
| PSEN/ | 1                  | 0          | 1                  | 0          |
| P0    | SFR data           | SFR data   | high-Z             | high-Z     |
| P1    | SFR data           | SFR data   | SFR data           | SFR data   |
| P2    | SFR data           | SFR data   | PCH                | SFR data   |
| P3    | SFR data           | SFR data   | SFR data           | SFR data   |

Figure 8. Status of Pins in Idle and Power Down Modes. "SFR data" means the port pins emit their internal register data. "PCH" is the high byte of the Program Counter.

There are two ways to terminate Idle. Activation of any enabled interrupt will cause the hardware to clear bit 0 of the PCON register, terminating the Idle mode. The interrupt will be serviced, and following RETI the next instruction to be executed will be the one following the instruction that invoked Idle.

The other way is with a hardware reset. Since the clock oscillator is still running, RST only needs to be held active for two machine cycles (24 oscillator periods) to complete the reset. Note that this exit from Idle writes 1s to all the ports, initializes all SFRs to their reset values, and restarts program execution from location 0.

**Power Down:** In the Power Down Mode ( $\overline{PD} = 0$  in Figure 7), the CPU puts the whole chip to sleep by turning off the oscillator. In case it was running from an external oscillator, it also gates off the path to the internal phase generators, so no internal clock is generated even if the external oscillator is still running. The on-chip RAM, however, saves its data, as long as  $V_{CC}$  is maintained. In this mode the only  $I_{CC}$  that flows is leakage, which is normally in the micro-amp range.

The Power Down Mode is invoked by setting bit 1 in the PCON register, using a byte instruction such as

```
ORL    PCON,#2
```

While the device is in Power Down, ALE and PSEN emit lows ( $V_{OL}$ ), as shown in Figure 8. The reason they are designed to emit lows is so that power can be removed from the rest of the circuit, if desired, while the 80C51BH is in its Power Down mode.

The port pins continue to emit whatever data was written to them. Note that Port 2 emits its P2 register data even if execution was from external program

memory. Port 0 also emits its P0 register data, but if execution was from external program memory, the P0 register data is FF. The oscillator is stopped, and the part remains in this state as long as  $V_{CC}$  is held, and until it receives an external reset signal.

The only exit from Power Down is a hardware reset. Since the oscillator was stopped, RST must be held active long enough for the oscillator to re-start and stabilize. Then the reset function initializes all the Special Function Registers (ports, timers, etc.) to their reset values, and re-starts the program from location 0. Therefore, timer reloads, interrupt enables, baud rates, port status, etc. need to be re-established. Reset does not affect the content of the on-chip data RAM. If  $V_{CC}$  was held during Power Down, the RAM data is still good.

### Using the Power Down Mode

The software-invoked Power Down feature offers a means of reducing the power consumption to a mere trickle in systems which are to remain dormant for some period of time, while retaining important data.

The user should give some thought to what state the port pins should be left in during the time the clock is stopped, and write those values to the port latches before invoking Power Down.

If  $V_{CC}$  is going to be held to the entire circuit, one would want to write values to the port latches that would deselect peripherals before invoking Power Down. For example, if external memory is being used, the P2 SFR should be loaded with a value which will not generate an active chip select to any memory device.

In some applications,  $V_{CC}$  to part of the system may be shut off during Power Down, so that even quiescent and standby currents are eliminated. Signal lines that connect to those chips must be brought to a logic low, whether the chip in question is CMOS, NMOS, or TTL, before  $V_{CC}$  is shut off to them. CMOS pins have parasitic pn junctions to  $V_{CC}$ , which will be forward biased if  $V_{CC}$  is reduced to zero while the pin is held at a logic high. NMOS pins often have FETs that look like diodes to  $V_{CC}$ . TTL circuits may actually be damaged by an input high if  $V_{CC} = 0$ . That's why the 80C51BH outputs lows at ALE and  $\overline{PSEN}$  during Power Down.

Figure 9 shows a circuit that can be used to turn  $V_{CC}$  off to part of the system during Power Down. The circuit will ensure that the secondary circuit is not de-energized until after the 80C31BH is in Power Down, and that the 80C31BH does not receive a reset (terminating the Power Down mode) before the secondary circuit is re-energized. Therefore, the program memory itself can be part of the secondary circuit.

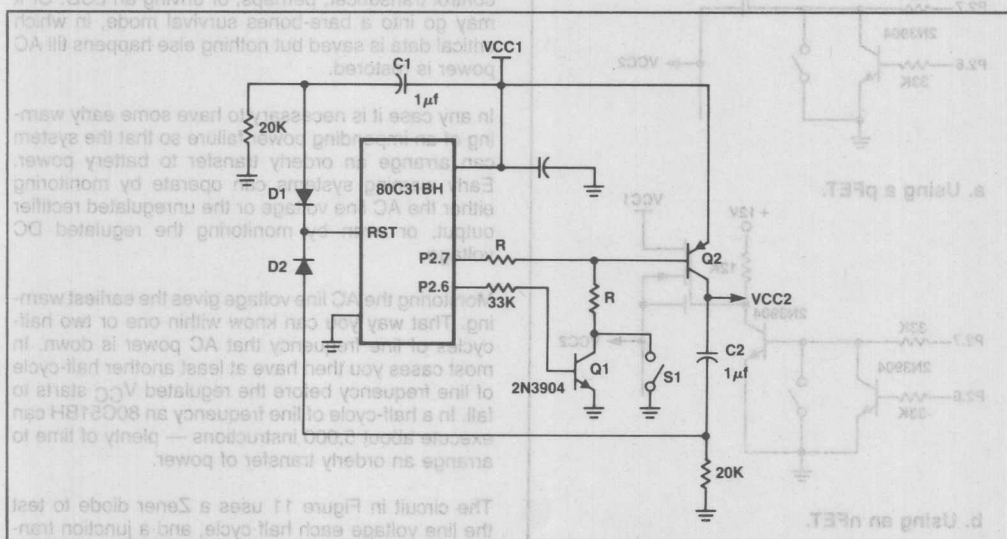


Figure 9. The 80C31BH de-energizes part of the circuit ( $V_{CC2}$ ) when it goes into Power Down. Selections of R and Q2 depend on  $V_{CC2}$  current draw.

intel®

80C31BH, capacitor C1 provides a power-on reset. The reset function writes 1s to all the port pins. The 1 at P2.6 turns Q1 on, enabling  $V_{CC}$  to the secondary circuit through transistor Q2. As the 80C31BH comes out of reset, Port 2 commences emitting the high byte of the Program Counter, which results in the P2.7 and P2.6 pins outputting 0s. The 0 at P2.7 ensures continuation of  $V_{CC}$  to the secondary circuit.

The system software must now write a 1 to P2.7 and a 0 to P2.6 in the Port 2 SFR, P2. These values will not appear at the Port 2 pins as long as the device is fetching instructions from external program memory. However, whenever the 80C31BH goes into Power Down, these values will appear at the port pins, and will shut off both transistors, disabling  $V_{CC}$  to the secondary circuit.

Closing the switch S1 re-energizes the secondary circuit, and at the same time sends a reset through C2 to the 80C31BH to wake it up. The diode D1 is to prevent C1 from hogging current from C2 during this secondary reset. D2 prevents C2 from discharging until after the 80C31BH is in Power Down.

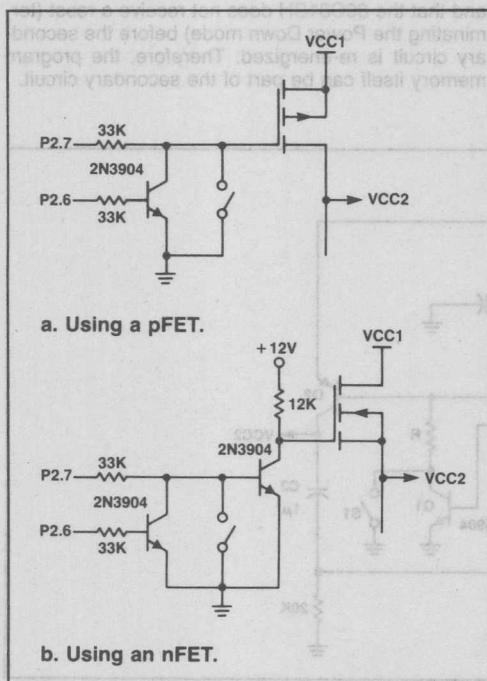


Figure 10. Using power MOSFETs to control  $V_{CC2}$ .

ing through the reset pin when  $V_{CC}$  to the secondary circuit goes to zero.

### Using Power MOSFETs to Control $V_{CC}$

Power MOSFETs are gaining in popularity (and availability). The easiest way to control  $V_{CC}$  is with a Logic Level pFET, as shown in Figure 10A. This circuit allows the full  $V_{CC}$  to be used to turn the device on. Unfortunately, power pFETs are not economically competitive with bipolar transistors of comparable ratings.

Power nFETs are both economical and available, and can be used in this application if a DC supply of higher voltage is available to drive the gate. Figure 10B shows how to implement a  $V_{CC}$  switch using a power nFET and a (nominally) +12V supply. The problem here is that if the device is on, its source voltage is +5V. To maintain the on state, the gate has to be another 5 or 10V above that. The "12V" supply is not particularly critical. A minimally filtered, unregulated rectifier will suffice.

### Battery Backup Systems

Here we consider circuits that normally draw power from the AC line, but switch to battery operation in the event of a power failure. We assume that in battery operation high-current loads will be allowed to die along with the AC power. The system may continue then with reduced functionality, monitoring a control transducer, perhaps, or driving an LCD. Or it may go into a bare-bones survival mode, in which critical data is saved but nothing else happens till AC power is restored.

In any case it is necessary to have some early warning of an impending power failure so that the system can arrange an orderly transfer to battery power. Early warning systems can operate by monitoring either the AC line voltage or the unregulated rectifier output, or even by monitoring the regulated DC voltage.

Monitoring the AC line voltage gives the earliest warning. That way you can know within one or two half-cycles of line frequency that AC power is down. In most cases you then have at least another half-cycle of line frequency before the regulated  $V_{CC}$  starts to fall. In a half-cycle of line frequency an 80C51BH can execute about 5,000 instructions — plenty of time to arrange an orderly transfer of power.

The circuit in Figure 11 uses a Zener diode to test the line voltage each half cycle, and a junction transistor to pass the information on to the 80C51BH. (Obviously a voltage comparator with a suitable reference source can perform the same function, if one

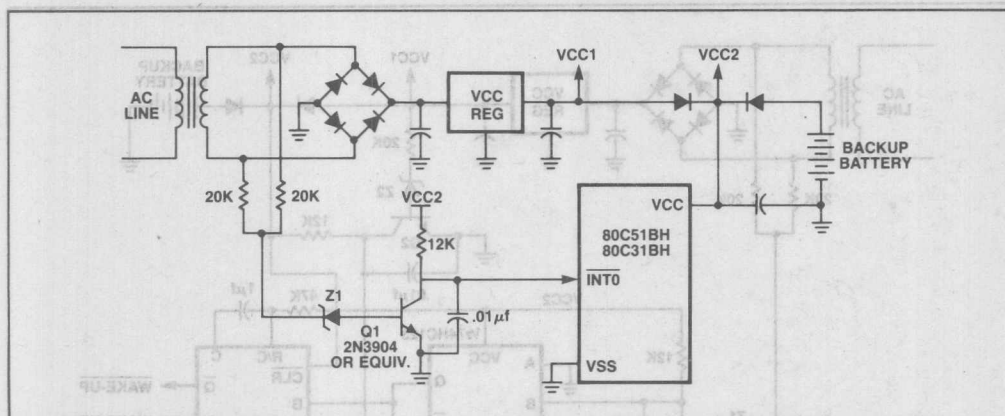


Figure 11. Power Failure Detector with Battery Backup. When AC power fails, VCC1 goes down and VCC2 is held.

prefers.) The way it works is if the line voltage reaches an acceptably high level, it breaks over Z1, drives Q1 to saturation, and interrupts the 80C51BH. The interrupt would be transition-activated, in this application. The interrupt service routine reloads one of the C51BH's timers to a value that will make it roll over in something between one and two half-cycles of line frequency. As long as the line voltage is healthy, the timer never rolls over, because it is reloaded every half cycle. If there is a single half cycle in which the line voltage doesn't reach a high enough level to generate the interrupt, the timer rolls over and generates a timer interrupt.

The timer interrupt then commences the transition to battery backup. Critical data needs to be copied into protected RAM. Signals to circuits that are going to lose power must be written to logic low. Protected circuits (those powered by VCC2) that communicate with unprotected circuits must be deselected. The microcontroller itself may be put into Idle, so that it can continue some level of interrupt-driven functionality, or it may be put into Power Down.

Note that if the CPU is going to invoke Power Down, the Special Function Registers may also need to be copied into protected RAM, since the reset that terminates the Power Down mode will also initialize all the SFRs to their reset values.

The circuit in Figure 11 does not show a wake-up mechanism. A number of choices are available, however. A pushbutton could be used to generate an interrupt, if the CPU is in Idle, or to activate reset, if the CPU is in Power Down.

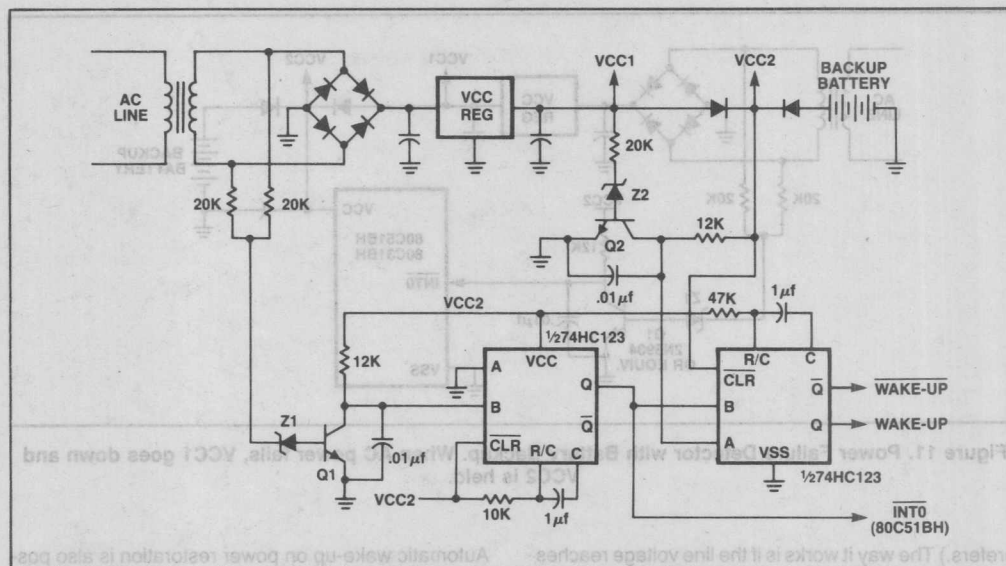
Automatic wake-up on power restoration is also possible. If the CPU is in Idle, it can continue to respond to any interrupts that might be generated by Q1. The interrupt service routine determines from the status of flag bits GF0 and GF1 in PCON that it is in Idle because there was a power outage. It can then sample VCC1 through a voltage comparator similar to Z1, Q1 in Figure 11. A satisfactory level of VCC1 would be indicated by the transistor being in saturation.

But perhaps you can't spare the timer that is the key to the operation of the circuit in Figure 11. In that case a retriggerable one-shot, triggered by the AC line voltage, can perform essentially the same function. Figure 12 shows an example of this type of power failure detector. A retriggerable one-shot (one half of a 74HC123) monitors the AC line voltage through transistor Q1. Q1 retriggers the one-shot every half cycle of line frequency. If the output pulse width is between one and two half-cycles of line frequency, then a single missing or low half cycle will generate an active low warning flag, which can be used to interrupt the microcontroller.

The interrupt routine takes care of the transition to battery back-up. From this point VCC1 may or may not actually drop out. The missing half-cycle of line voltage that caused the power down sequence may have been nothing more than a short glitch. If the AC line comes back strong enough to trigger the one-shot while VCC1 is still up (as indicated by the state of transistor Q2), then the other half of the 74HC123 will generate a wake-up signal.

Having been awakened, the 80C51BH will stay





**Figure 12. Power Failure Detector uses retriggerable one-shots to flag impending power outage and generate automatic wake-up when power returns.**

awake for at least another half-cycle of line frequency (another 5,000 or so instructions) before possibly being told to arrange another transfer of power. Consequently, if the line voltage is jittering erratically around the switchover point (determined by diode Z1), the system will limp along executing in half-cycle units of line frequency.

On the other hand, if the power outage is real and lengthy,  $V_{CC1}$  will eventually fall below the level at which the backup battery takes over. The backup battery maintains power to the 80C51BH, and to the 74HC123, and to whatever other circuits are being protected during this outage. The battery voltage must be high enough to maintain  $V_{CCMIN}$  specs to the 80C51BH.

If the microcontroller is an 80C31BH, executing out of external ROM, and if the C31BH is put into Idle during the power outage, then the external ROM must also be supplied by the battery. On the other hand, if the C31BH is put into Power Down during the outage, then the ROM can be allowed to die with the AC power. The considerations here are the same as in Figure 9:  $V_{CC}$  to the ROM is still up at the time Power Down is invoked, and we must ensure (through selection of diode Z2 in Figure 12) that the 80C31BH is not awakened till ROM power is back in spec.

## Power Switchover Circuits

Battery backup systems need to have a way for the protected circuits to draw power from the line-operated power supply when that source is available, and to switch over to battery power when required. The switchover circuit is simple if the entire system is to be battery powered in the event of a line power outage. In that case a pair of diodes suffice, as shown in Figure 12, provided  $V_{CCMIN}$  specs are still met after the diode drop has been subtracted from its respective power source.

The situation becomes more complicated when part of the circuit is going to be allowed to die when the AC power goes out. In that case it is difficult to maintain equal  $V_{CCS}$  to protected and unprotected circuits (and possibly dangerous not to).

The problem can be alleviated by using a Schottky diode instead of a 1N4001, for its lower forward voltage drop. The 1N5820, for example, has a forward drop of about 0.35V at 1A.

Other solutions are to use a transistor or power MOSFET switch, as shown in Figure 13. With minor modifications this switch can be controlled by port pins.

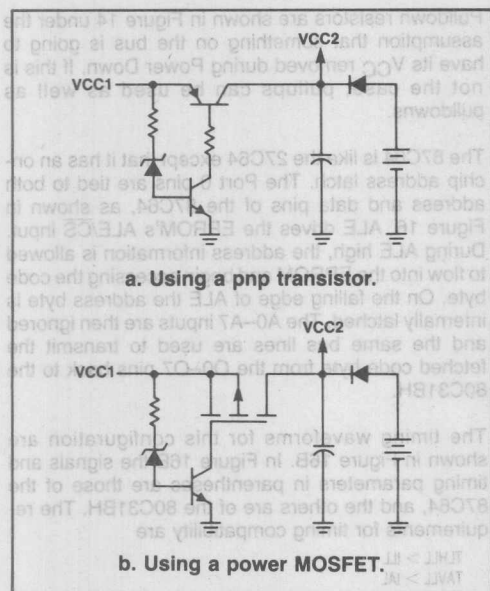


Figure 13. Power Switchover Ckts.

### 80C31BH + CHMOS EPROM

The 27C64 and 87C64 are Intel's 8K byte CHMOS EPROMs. The 27C64 requires an external address latch, and can be used with the 80C31BH as shown in Figure 14A. In most 8031 + 2764 (HMOS) appli-

cations, the 2764's Chip Enable ( $\overline{CE}$ ) pin is hard-wired to ground (since it's normally the only program memory on the bus). This can be done with the CHMOS versions as well, but there is some advantage in connecting  $\overline{CE}$  to ALE, as shown in Figure 14. The advantage is that if the 80C31BH is put into Idle mode, since ALE goes to a 1 in that mode, the 27C64 will be deselected and go into a low current standby mode.

The timing waveforms for this configuration are shown in Figure 14B. In Figure 14B the signals and timing parameters in parentheses are those of the 27C64, and the others are of the 80C31BH, except  $T_{prop}$  is a parameter of the address latch. The requirements for timing compatibility are

TAVIV -  $T_{prop} > t_{ACC}$   
 TLLIV -  $t_{CE}$   
 TPLIV -  $t_{OE}$   
 TPIXZ -  $t_{DF}$

If the application is going to use the Power Down mode then we have another consideration: In Idle,  $ALE = \overline{PSEN} = 1$ , and in Power Down,  $ALE = \overline{PSEN} = 0$ . In a realistic application there are likely to be more chips in the circuit than are shown in Figure 14, and it is likely that the nonessential ones will have their  $V_{CC}$  removed while the CPU is in Power Down. In that case the EPROM and the address latch should be among the chips that have  $V_{CC}$  removed, and logic lows are exactly what are required at ALE and  $\overline{PSEN}$ .

But if  $V_{CC}$  is going to be maintained to the EPROM during Power Down, then it will be necessary to de-

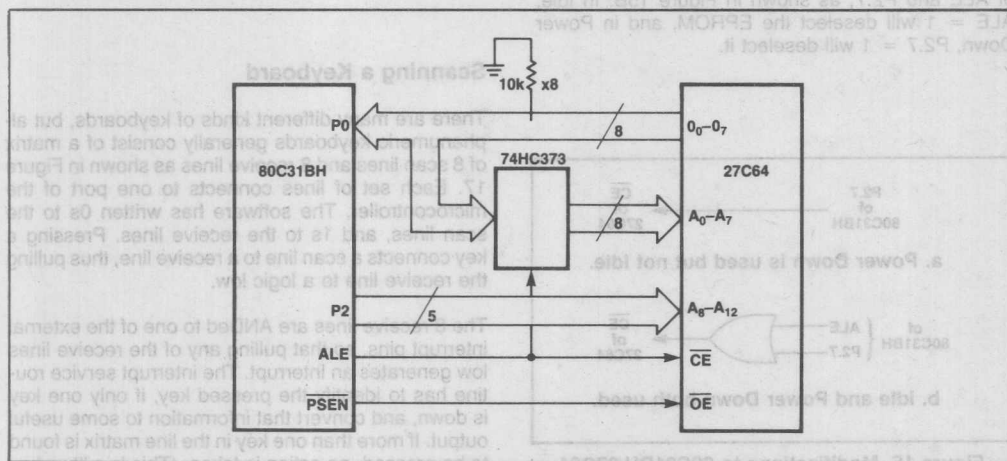


Figure 14a. 80C31BH + 27C64.

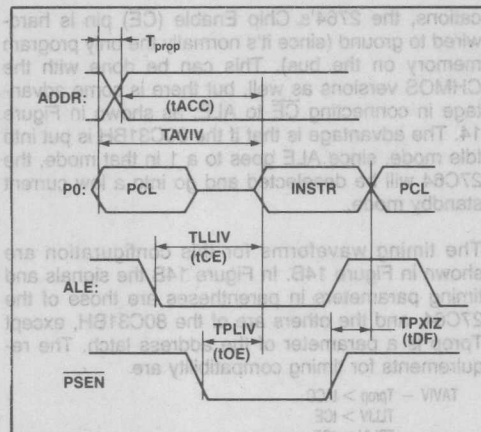


Figure 14b. Timing Waveforms for 80C31BH + 27C64.

If the application is going to use the Power Down mode then we have another consideration: in Idle,  $\overline{CE} = \overline{PSEN} = 1$  and in Power Down,  $\overline{ALE} = \overline{PSEN} = 1$ . select the EPROM when the CPU is in Power Down. If Idle is never invoked,  $\overline{CE}$  of the EPROM can be connected to P2.7 of the 80C31BH, as shown in Figure 15A. In normal operation, P2.7 will be emitting the MSB of the Program Counter, which is 0 if the program contains less than 32K of code. Then when the CPU goes into Power Down, the Port 2 pins emit P2 SFR data, which puts a 1 at P2.7, thus deselecting the EPROM.

If Idle and Power Down are both going to be used,  $\overline{CE}$  of the EPROM can be driven by the logical OR of ALE and P2.7, as shown in Figure 15B. In Idle,  $\overline{ALE} = 1$  will deselect the EPROM, and in Power Down,  $P2.7 = 1$  will deselect it.

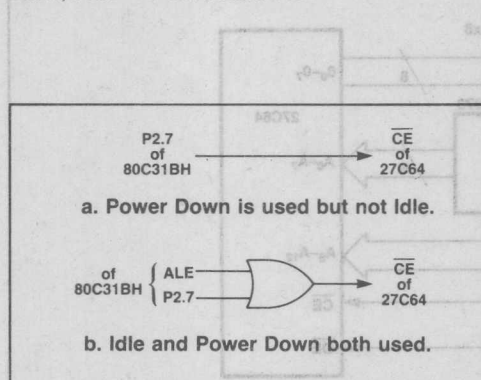


Figure 15. Modifications to 80C31BH/27C64 interface.

Pulldown resistors are shown in Figure 14 under the assumption that something on the bus is going to have its  $V_{CC}$  removed during Power Down. If this is not the case, pullups can be used as well as pulldowns.

The 87C64 is like the 27C64 except that it has an on-chip address latch. The Port 0 pins are tied to both address and data pins of the 87C64, as shown in Figure 16. ALE drives the EPROM's  $\overline{ALE}/\overline{CS}$  input. During ALE high, the address information is allowed to flow into the EPROM and begin accessing the code byte. On the falling edge of ALE the address byte is internally latched. The A0-A7 inputs are then ignored and the same bus lines are used to transmit the fetched code byte from the O0-O7 pins back to the 80C31BH.

The timing waveforms for this configuration are shown in Figure 16B. In Figure 16B the signals and timing parameters in parentheses are those of the 87C64, and the others are of the 80C31BH. The requirements for timing compatibility are

TLHL > ILL  
TAVL > IAL  
TLLAX > ILA  
TLLIV > IACL  
TPLIV > IOE  
TLLPL > ICOE  
TPXIZ > IOHZ

The same considerations apply to the 87C64 as to the 27C64 with regards to the Idle and Power Down modes. Basically you want  $\overline{CS} = 1$  if  $V_{CC}$  is maintained to the EPROM, and  $\overline{CS} = \overline{OE} = 0$  if  $V_{CC}$  is removed.

## Scanning a Keyboard

There are many different kinds of keyboards, but alphanumeric keyboards generally consist of a matrix of 8 scan lines and 8 receive lines as shown in Figure 17. Each set of lines connects to one port of the microcontroller. The software has written 0s to the scan lines, and 1s to the receive lines. Pressing a key connects a scan line to a receive line, thus pulling the receive line to a logic low.

The 8 receive lines are ANDed to one of the external interrupt pins, so that pulling any of the receive lines low generates an interrupt. The interrupt service routine has to identify the pressed key, if only one key is down, and convert that information to some useful output. If more than one key in the line matrix is found to be pressed, no action is taken. (This is a "two key lock-out" scheme.)

On some keyboards, certain keys (Shift, Control, Escape, etc.) are not a part of the line matrix. These keys would connect directly to a port pin on the microcontroller, and would not cause lock-out if pressed simultaneously with a matrix key, nor generate an interrupt if pressed singly.

Normally the microcontroller would be in Idle mode when a key has not been pressed, and another task is not in progress. Pressing a matrix key generates

an interrupt, which terminates the Idle. The interrupt service routine would first call a 30 msec (or so) delay to debounce the key, and then set about the task of identifying which key is down.

First, the current state of the receive lines is latched into an internal register. If a single key is down, all but one of these lines would be read as 1s. Then 0s are written to the receive lines and 1s to the scan lines, and the scan lines are read. If a single key is

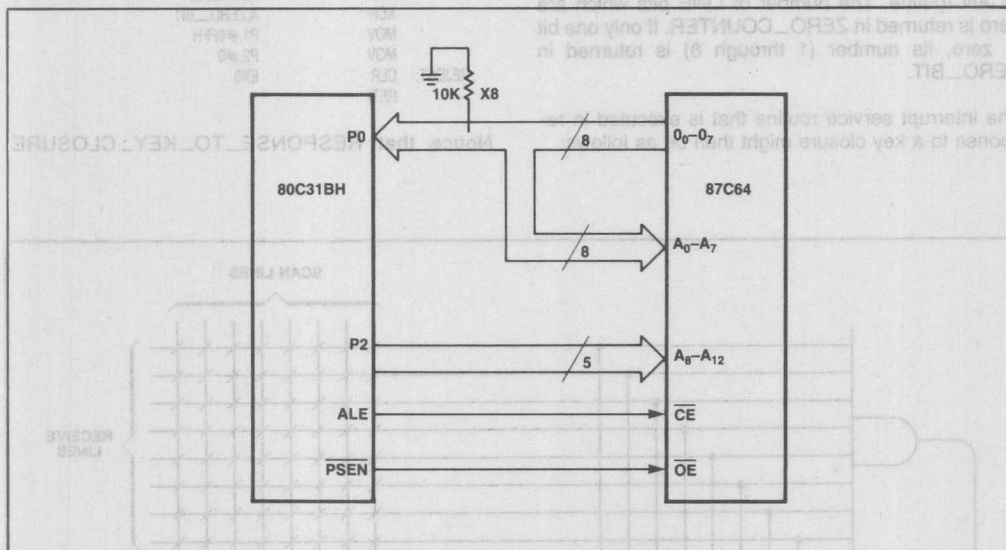


Figure 16a. 80C31BH + 87C64.

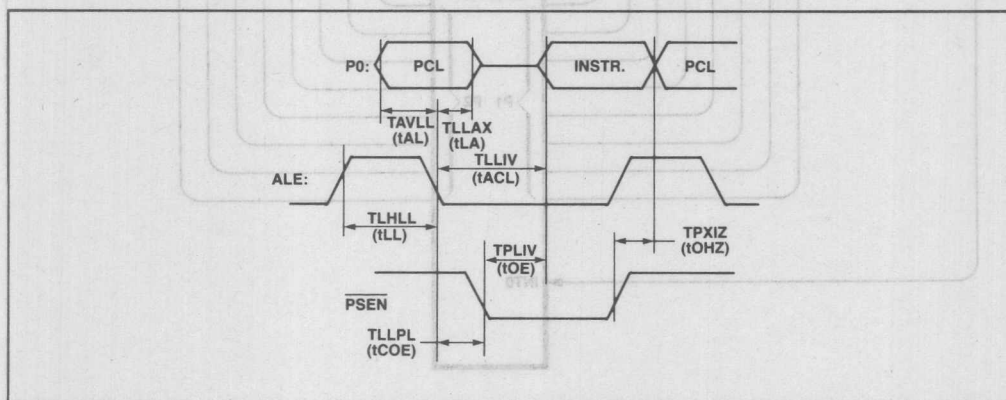


Figure 16b. Timing Waveforms for 80C31BH + 87C64.



down, all but one of these lines would be read as 1s. By locating the single 0 in each set of lines, the pressed key can be identified. If more than one matrix key is down, one or both sets of lines will contain multiple 0s.

A subroutine is used to determine which of 8 bits in either set of lines is 0, and whether more than one bit is 0. Figure 18 shows a subroutine (SCAN) which does that using the 8051's bit-addressing capability. To use the subroutine, move the line data into a bit-addressable RAM location named LINE, and call the SCAN routine. The number of LINE bits which are zero is returned in ZERO\_COUNTER. If only one bit is zero, its number (1 through 8) is returned in ZERO\_BIT.

The interrupt service routine that is executed in response to a key closure might then be as follows:

```

RESPONSE__TO__KEY__CLOSURE:
CALL    DEBOUNCE__DELAY
MOV     LINE,P1; ;See Figure 17.
CALL    SCAN
DJNZ    ZERO_COUNTER,REJECT
MOV     ADDRESS,ZERO_BIT
MOV     P2,#0FFH; ;See Figure 17.
MOV     P1,#0
MOV     LINE,P2
CALL    SCAN
DJNZ    ZERO_COUNTER,REJECT
XCH     A,ZERO_BIT
SWAP    A
ORL     ADDRESS,A
XCH     A,ZERO_BIT
MOV     P1,#0FFH
MOV     P2,#0
CLR     EX0
RETI
    
```

Notice that RESPONSE\_\_TO\_\_KEY\_\_CLOSURE

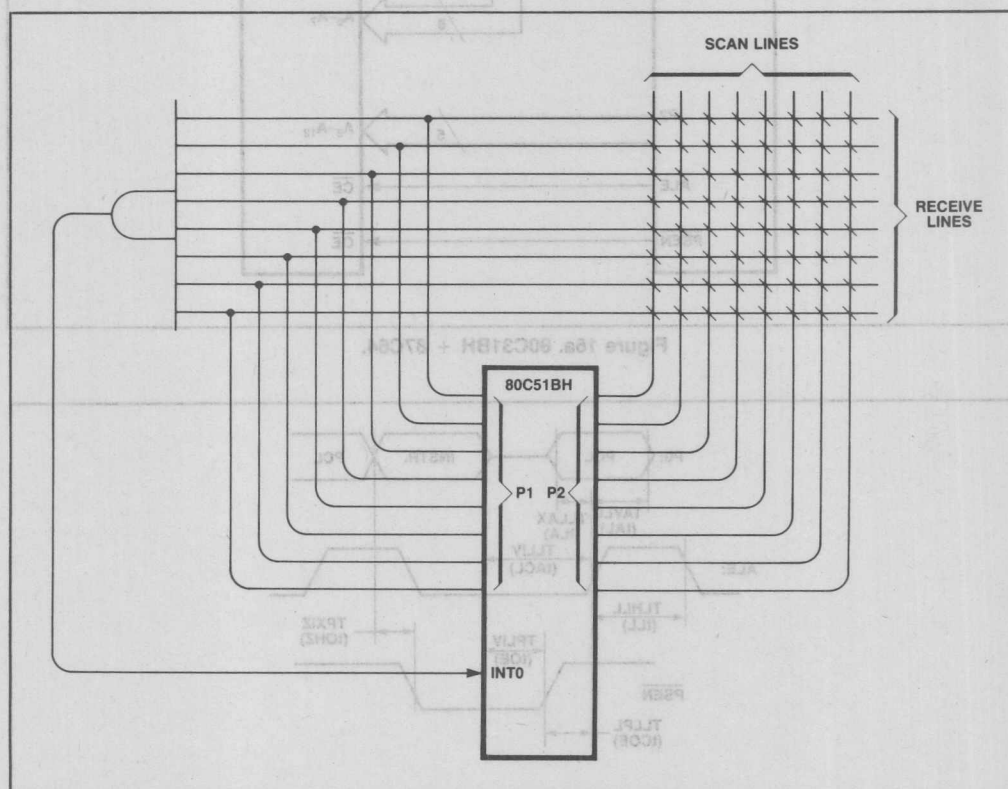


Figure 18. Scanning a Keyboard.

```

SCAN:  MOV     ZERO_COUNTER,#0 ; ZERO_COUNTER counts the number of 0s in LINE.
        JB     LINE.0,ONE      ; Test LINE bit 0.
        INC     ZERO_COUNTER    ; If LINE.0 = 0, increment ZERO_COUNTER
        MOV     ZERO_BIT,#1     ; and record that line number 1 is active.
ONE:    JB     LINE.1,TWO      ; Procedure continues for other LINE bits.
        INC     ZERO_COUNTER
        MOV     ZERO_BIT,#2     ; Line number 2 is active.
TWO:    JB     LINE.2,THREE
        INC     ZERO_COUNTER
        MOV     ZERO_BIT,#3     ; Line number 3 is active.
THREE:  JB     LINE.3,FOUR
        INC     ZERO_COUNTER
        MOV     ZERO_BIT,#4     ; Line number 4 is active.
FOUR:   JB     LINE.4,FIVE
        INC     ZERO_COUNTER
        MOV     ZERO_BIT,#5     ; Line number 5 is active.
FIVE:   JB     LINE.5,SIX
        INC     ZERO_COUNTER
        MOV     ZERO_BIT,#6     ; Line number 6 is active.
SIX:    JB     LINE.6,SEVEN
        INC     ZERO_COUNTER
        MOV     ZERO_BIT,#7     ; Line number 7 is active.
SEVEN:  JB     LINE.7,EIGHT
        INC     ZERO_COUNTER
        MOV     ZERO_BIT,#8     ; Line number 8 is active.
EIGHT:  RET

```

Figure 18. Subroutine SCAN determines which of 8 bits in LINE is zero.

does not change the Accumulator, the PSW, nor any of the registers R0 through R7. Neither do SCAN or DEBOUNCE\_DELAY.

What we come out with then is a one-byte key address (ADDRESS) which identifies the pressed key. The key's scan line number is in the upper nibble of ADDRESS, and its receive line number is in the lower nibble. ADDRESS can be used in a look-up table to generate a key code to transmit to a host computer, and/or to a display device.

The keyboard interrupt itself must be edge-triggered, rather than level-activated, so that the interrupt routine is invoked when a key is pressed, and is not constantly being repeated as long as the key is held down. In edge-triggered mode, the on-chip hardware clears the interrupt flag (EX0, in this case) as the service routine is being vectored to. In this application, however, contact bounce will cause several more edges to occur after the service routine has been vectored to, during the DEBOUNCE\_DELAY routine. Consequently it is necessary to clear EX0 again in software before executing RETI.

The debounce delay routine also takes advantage of the Idle mode. In this routine a timer must be preloaded with a value appropriate to the desired length of delay. This value would be

$$\text{timer preload} = \frac{(\text{osc kHz}) \times (\text{delay time msec})}{12}$$

For example, with a 3.58MHz oscillator frequency, a 30 msec delay could be obtained using a preload value of -8950, or DD0A, in hex digits.

In the debounce delay routine (Figure 19), the timer interrupt is enabled and set to a higher priority than the keyboard interrupt, because as we invoke Idle, the keyboard interrupt is still "in progress." An interrupt of the same priority will not be acknowledged, and will not terminate the Idle mode. With the timer interrupt set to priority 1, while the keyboard interrupt is a priority 0, the timer interrupt, when it occurs, will be acknowledged and will wake up the CPU. The timer interrupt service routine does not itself have to do anything. The service routine might be nothing more than a single RETI instruction. RETI from the timer interrupt service routine then returns execution to the debounce delay routine, which shuts down the timer and returns execution to the keyboard service routine.

### Driving an LCD

An LCD (Liquid Crystal Display) consists of a backplane and any number of segments or dots which will be used to form the image being displayed. Applying a voltage (nominally 4 or 5V) between any segment and the backplane causes the segment to darken. The only catch is that the polarity of the applied voltage has to be periodically reversed, or else a chem-

```

DEBOUNCE_DELAY:
    MOV     TL1,#TL1_PRELOAD ; Preload low byte.
    MOV     TH1,#TH1_PRELOAD ; Preload high byte.
    SETB    ET1               ; Enable Timer 1 interrupt.
    SETB    PT1               ; Set Timer 1 interrupt to high priority.
    SETB    TR1               ; Start timer running.
    ORL     PCON,#1           ; Invoke Idle mode.
    ; The next instruction will not be executed until the delay times out.
    CLR     TR1               ; Stop the timer.
    CLR     PT1               ; Back to priority 0 (if desired).
    CLR     ET1               ; Disable Timer 1 interrupt (if desired).
    RET

```

Figure 19. Subroutine DEBOUNCE\_DELAY puts the 80C51BH into Idle during the delay time.

ical reaction takes place in the LCD which causes deterioration and eventual failure of the liquid crystal.

To prevent this happening, the backplane and all the segments are driven with an AC signal, which is derived from a rectangular voltage waveform. If a segment is to be "off" it is driven by the same waveform as the backplane. Thus it is always at backplane potential. If the segment is to be "on" it is driven with a waveform that is the inverse of the backplane waveform. Thus it has about 5V of periodically changing polarity between it and the backplane.

With a little software overhead, the 80C51BH can perform this task without the need for additional LCD drivers. The only drawback is that each LCD segment uses up one port pin, and the backplane uses one more. If more than, say, two 7-segment digits are being driven, there aren't many port pins left for other tasks. Nevertheless, assuming a given application leaves enough port pins available to support this task, the considerations for driving the LCD are as follows.

Suppose, for example, it is a 2-digit display with a decimal point. One port (TENS\_DIGIT) connects to the 7 segments of the tens digit plus the backplane. Another port (ONES\_DIGIT) connects to a decimal point plus the 7 segments of the ones digit.

One of the 80C51BH's timers is used to mark off half-periods of the drive voltage waveform. The LCD drive waveform should have a rep rate between 30 and 100 Hz, but it's not very critical. A half-period of 12 msec will set the rep rate to about 42 Hz. The preload/reload value to get 12 msec to rollover is the 2's complement negative of the oscillator frequency in kHz: If the oscillator frequency is 3.58MHz, the reload value is -3580, or F204 in hex digits.

Now, the 80C51BH would normally be in Idle, to conserve power, during the time that the LCD and other

tasks are not requiring servicing. When the timer rolls over it generates an interrupt, which brings the 80C51BH out of Idle. The service routine reloads the timer (for the next rollover), and inverts the logic levels of all the pins that are connected to the LCD. It might look like this:

```

LCD_DRIVE_INTERRUPT:
    MOV     TL1,#LOW(-XTAL_FREQ)
    MOV     TH1,#HIGH(-XTAL_FREQ)
    XRL     TENS_DIGIT,#0FFH
    XRL     ONES_DIGIT,#0FFH
    RETI

```

To update the display, one would use a look-up table to generate the characters. In the table, "on" segments are represented as 1s, and "off" segments as 0s. The backplane bit is represented as a 0. The quantity to be displayed is stored in RAM as a BCD value. The look-up table operates on the low nibble of the BCD value, and produces the bit pattern that is to be written to either the ones digit or the tens digit. Before the new patterns can be written to the LCD, the LCD drive interrupt has to be disabled. That is to prevent a polarity reversal from taking place between the times the two digits are written. An update subroutine is shown in Figure 20.

### Using an LCD Driver

As was noted, driving an LCD directly with an 80C51BH uses a lot of port pins. LCD drivers are available in CMOS to interface an 80C51BH to a 4-digit display using only 7 of the C51BH's I/O pins. Basically, the C51BH tells the LCD driver what digit is to be displayed (4 bits) and what position it is to be displayed in (2 bits), and toggles a Chip Select pin to tell the driver to latch this information. The LCD driver generates the display characters (hex digits), and takes care of the polarity reversals using its own RC oscillator to generate the timing.

Figure 25 shows an 80C51BH working with an ICM7211M to drive a 4-digit LCD, and the software that updates the display. The time of 128 msec. The frequency count would be 128 msec x 9KHz = 1152. One could equally well send information to the LCD driver over the bus. In that case, one would set up the Accumulator with the digit select and data input bits, and execute a MOVX @ R0,A instruction. The LCD driver's chip select would be driven by the CPU's WR signal. This is a little easier in software than the direct bit manipulation shown in Figure 21. However, it uses more I/O pins, unless there is already some external memory involved. In that case, no extra pins are used up by adding the LCD driver to the bus.

### Resonant Transducers

Analog transducers are often used to convert the value of a physical property, such as temperature, pressure, etc., to an analog voltage. These kinds of transducers then require an analog-to-digital converter to put the measurement into a form that is compatible with a digital control system. Another kind of transducer is now becoming available that encodes the value of the physical property into a signal that can be directly read by a digital control system. These devices are called resonant transducers.

Resonant transducers are oscillators whose frequency depends in a known way on the physical property being measured. These devices output a train of rectangular pulses whose repetition rate encodes the value of the quantity being measured. The pulses can in most cases be fed directly into the 80C51BH, which then measures either the frequency or period of the incoming signal, basing the measurement on the accuracy of its own clock oscillator. The 80C51BH can even do this in its sleep; that is, in Idle.

When the frequency or period measurement is completed, the C51BH wakes itself up for a very short time to perform a sanity check on the measurement and convert it in software to any scaling of the measured quantity that may be desired. The software conversion can include corrections for nonlinearities in the transducer's transfer function.

Resolution is also controlled by software, and can even be dynamically varied to meet changing needs as a situation becomes more critical. For example, in a process controller you can increase your resolution ("fine tune" the control, as it were) as the process approaches its target.

The nominal reference frequency of the output signal from these devices is in the range of 20Hz to 500kHz, depending on the design. Transducers are available that have a full scale frequency shift of 2 to 1. The transducer operates from a supply voltage range of 3V to 20V, which means it can operate from the same supply voltage as the 80C51BH. At 5V, the transducer draws less than 5 mA (reference 7). It can normally be connected directly to one of the C51BH's port pins, as shown in Figure 22.

### Frequency Measurements

Measuring a frequency means counting pulses for a known sample time. Two timer/counters can be used, one to mark off the sample time and one to count pulses. If the frequency being counted doesn't exceed 50kHz or so, one may equally well connect the transducer signal to one of the external interrupt pins, and count pulses in software. That frees up one timer, with very little cost in CPU time.

The count that is directly obtained is TxF, where T is the sample time and F is the frequency. The full scale

```

UPDATE_LCD:  ORG 0000H ; Select fourth digit (address = 0000H)
              CLR     ET1 ; Disable LCD drive interrupt
              MOV     DPTR, #TABLE_ADDRESS ; Look-up table begins at TABLE_ADDRESS
              MOV     A, BCD_VALUE ; Digits to be displayed.
              SWAP    A ; Move tens digit to low nibble
              ANL     A, #0FH ; Mask off high nibble
              MOVX   C, @A+DPTR ; Tens digit pattern to accumulator.
              MOV     TENS_DIGIT, A ; Update LCD tens digit.
              MOV     A, BCD_VALUE ; Digits to be displayed.
              ANL     A, #0FH ; Mask off tens digit.
              MOVX   C, @A+DPTR ; Ones digit pattern to accumulator.
              MOV     C, DECIMAL_POINT ; Add decimal point to segment
              MOV     ACC, 7 ; pattern. Update LCD decimal point
              MOV     ONES_DIGIT, A ; and ones digit
              SETB    ET1 ; Re-enable LCD drive interrupt.
              RET
    
```

Figure 20. UPDATE\_LCD routine writes two digits to an LCD.



range is  $T_x(F_{max}-F_{min})$ . For n-bit resolution

$$1 \text{ LSB} = \frac{T_x(F_{max}-F_{min})}{2^n}$$

Therefore the sample time required for n-bit resolution is

$$T = \frac{2^n}{F_{max}-F_{min}}$$

For example, 8-bit resolution in the measurement of

a frequency that varies between 7kHz and 9kHz would require, according to this formula, a sample time of 128 msec. The maximum acceptable frequency count would be  $128 \text{ msec} \times 9\text{kHz} = 1152$  counts. The minimum would be 896 counts. Subtracting 896 from each frequency count (or presetting the frequency counter to  $-896 = 0FC80H$ ) would allow the frequency to be reported on a scale of 0 to FF in hex digits.

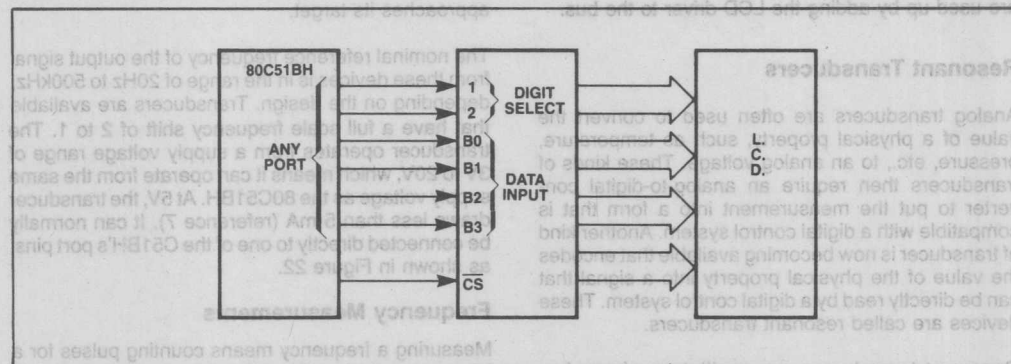


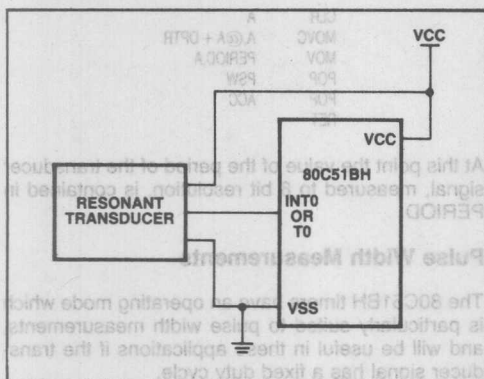
Figure 21a. Using an LCD driver.

```

UPDATE_LCD:
MOV     A, DISPLAY_HI      ; High byte of 4-digit display.
SETB    DIGIT_SELECT_2     ; Select leftmost digit of LCD.
SETB    DIGIT_SELECT_1     ; (Digit address = 11B.)
CALL    SHIFT_AND_LOAD     ; High nibble of high byte to selected digit.
CLR     DIGIT_SELECT_1     ; Select second digit of LCD (address = 10B).
CALL    SHIFT_AND_LOAD     ; Low nibble of high byte to selected digit.
MOV     A, DISPLAY_LO      ; Low byte of 4-digit display.
CLR     DIGIT_SELECT_2     ; Select third digit of LCD.
SETB    DIGIT_SELECT_1     ; (Digit address = 01B.)
CALL    SHIFT_AND_LOAD     ; High nibble of low byte to selected digit.
CLR     DIGIT_SELECT_1     ; Select fourth digit (address = 00B).
CALL    SHIFT_AND_LOAD     ; Low nibble of low byte to selected digit.
RET

SHIFT_AND_LOAD:
RLC     A                   ; MSB to carry bit (CY).
MOV     DATA_INPUT_B3, C   ; CY to Data Input pin B3.
RLC     A                   ; Next bit to CY.
MOV     DATA_INPUT_B2, C   ; CY to Data Input pin B2.
RLC     A                   ; Next bit to CY.
MOV     DATA_INPUT_B1, C   ; CY to Data Input pin B1.
RLC     A                   ; Last bit to CY.
MOV     DATA_INPUT_B0, C   ; CY to Data Input pin B0.
CLR     CHIP_SELECT         ; Toggle Chip Select.
SETB    CHIP_SELECT         ; 0-to-1 transition latches info.
RET
    
```

Figure 21b. UPDATE\_LCD routine writes 4 digits to an LCD driver.



**Figure 22. Resonant Transducer does not require an A/D converter.**

To implement the measurement, one timer is used to establish the sample time. The timer is preset to a value that causes it to roll over at the end of the sample time, generating an interrupt and waking the CPU from its Idle mode. The required preset value is the 2's complement negative of the sample time measured in machine cycles. The conversion from sample time to machine cycles is to multiply it by 1/12 the clock frequency. For example, if the clock frequency is 12MHz, then a sample time of 128 msec is

$$(128 \text{ msec}) \times (12000\text{kHz}) / 12 = 128000 \text{ machine cycles.}$$

Then the required preset value to cause the timer to roll over in 128 msec is

$$-128000 = \text{FE0C00, in hex digits.}$$

Note that the preset value is 3 bytes wide, whereas the timer is only 2 bytes wide. This means the timer must be augmented in software in the timer interrupt routine to three bytes. The 80C51BH has a DJNZ instruction (decrement and jump if not zero) that makes it easier to code the third timer byte to count down instead of up. If the third timer byte counts down, its reload value is the 2's complement of what it would be for an up-counter. For example, if the 2's complement of the sample time is FE0C00, then the reload value for the third timer byte would be 02, instead of FE. The timer interrupt routine might then be:

```

TIMER_INTERRUPT_ROUTINE:
    DJNZ THIRD_TIMER_BYTE, OUT
    MOV TLO, #0
    MOV TH0, #0CH
    MOV THIRD_TIMER_BYTE, #2
    MOV FREQUENCY_COUNTER_LO

```

```

;Preset COUNTER to - 896: 0
MOV COUNTER_LO, #80H
MOV COUNTER_HI, #0FCH
OUT: RETI

```

At this point the value of the frequency of the transducer signal, measured to 8 bit resolution, is contained in FREQUENCY. Note that the timer can be reloaded on the fly. Note too that for 8-bit resolution only the low byte of the frequency counter needs to be read, since the high byte is necessarily 0. However, one may want to test the high byte to ensure that it is zero, as a sanity check on the data. Both bytes, of course must be reloaded.

## Period Measurements

Measuring the period of the transducer signal means measuring the total elapsed time over a known number, N, of transducer pulses. The quantity that is directly measured is NT, where T is the period of the transducer signal in machine cycles. The relationship between T in machine cycles and the transducer frequency F in arbitrary frequency units is

$$T = \frac{F_{\text{xtal}}}{F} \times (1/12),$$

where Fxtal is the 80C51BH clock frequency, in the same units as F.

The full scale range then is Nx(Tmax-Tmin). For n-bit resolution

$$1 \text{ LSB} = \frac{N \times (T_{\text{max}} - T_{\text{min}})}{2^n}$$

Therefore the number of periods over which the elapsed time should be measured is

$$N = \frac{2^n}{T_{\text{max}} - T_{\text{min}}}$$

However, N must also be an integer. It is logical to evaluate the above formula (don't forget Tmax and Tmin have to be in machine cycles) and select for N the next higher integer. This selection gives a period measurement that has somewhat more than n-bit resolution, but it can be scaled back if desired.

For example, suppose we want 8-bit resolution in the measurement of the period of a signal whose frequency varies from 7.1kHz to 9kHz. If the clock frequency is 12MHz, then Tmax is (12000kHz/7.1kHz)x(1/12) = 141 machine cycles. Tmin is 111 machine cycles. The required value for N, then, is 256/(141-111) = 8.53 periods, according to the formula. Using N = 9 periods will give a maximum NT value of 141x9 = 1269 machine cycles. The minimum NT will be 111x9 = 999 machine cycles. A lookup table can be used to scale these

values back to a range of 0 to 255, giving precisely the 8-bit resolution desired.

To implement the measurement, one timer is used to measure the elapsed time, NT. The transducer is connected to one of the external interrupt pins, and this interrupt is configured to the transition-activated mode. In the transition-activated mode every 1-to-0 transition in the transducer output will generate an interrupt. The interrupt routine counts transducer pulses, and when it gets to the predetermined N, it reads and clears the timer. For the specific example cited above, the interrupt routine might be:

```

INTERRUPT_RESPONSE:
    DJNZ N,OUT
    MOV N,#9
    CLR EA
    CLR TR1
    MOV NT_LO,TL1
    MOV NT_HI,TH1
    MOV TL1,#9
    MOV TH1,#0
    SETB TR1
    SETB EA
    CALL LOOKUP_TABLE
    OUT: RETI

```

In this routine a pulse counter N is decremented from its preset value, 9, to zero. When the counter gets to zero it is reloaded to 9. Then all interrupts are blocked for a short time while the timer is read and cleared. The timer is stopped during the read and clear operations, so "clearing" it actually means presetting it to 9, to make up for the 9 machine cycles that are missed while the timer is stopped.

The subroutine LOOKUP\_TABLE is used to scale the measurement back to the desired 8-bit resolution. It can also include built-in corrections for errors or nonlinearities in the transducer's transfer function.

The subroutine uses the MOV A, @A+DPTR instruction to access the table, which contains 270 entries commencing at the 16-bit address referred to as TABLE. The subroutine must compute the address of the table entry that corresponds to the measured value of NT. This address is

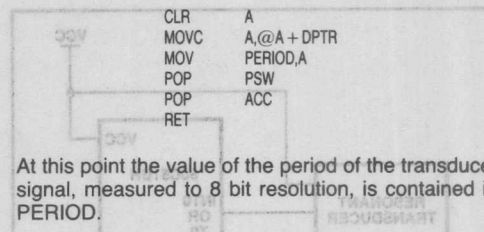
$$DPTR = TABLE + NT - NTMIN,$$

where NTMIN = 999, in this specific example.

```

LOOKUP_TABLE:
    PUSH ACC
    PUSH PSW
    MOV A,#LOW(TABLE-NTMIN)
    ADD A,NT_LO
    MOV DPL,A
    MOV A,#HIGH(TABLE-NTMIN)
    ADDC A,NT_HI
    MOV DPH,A

```



At this point the value of the period of the transducer signal, measured to 8 bit resolution, is contained in PERIOD.

## Pulse Width Measurements

The 80C51BH timers have an operating mode which is particularly suited to pulse width measurements, and will be useful in these applications if the transducer signal has a fixed duty cycle.

In this mode the timer is turned on by the on-chip circuitry in response to an input high at the external interrupt pin, and off by an input low, and it can do this while the 80C51BH is in Idle. (The "GATE" mode of timer operation is described in the Intel Microcontroller Handbook.) The external interrupt itself can be enabled, so the same 1-to-0 transition from the transducer that turns off the timer also generates an interrupt. The interrupt routine then reads and resets the timer.

The advantage of this method is that the transducer signal has direct access to the timer gate, with the result that variations in interrupt response time have no effect on the measurement.

Resonant transducers that are designed to fully exploit the GATE mode have an internal divide-by-N circuit that fixes the duty cycle at 50% and lowers the output frequency to the range of 250 to 500 Hz (to control RFI). The transfer function between transducer period and measurand value is approximately linear, with known and repeatable error functions.

## HMOS/CHMOS Interchangeability

The CHMOS version of the 8051 is architecturally identical with the HMOS version, but there are nevertheless some important differences between them which the designer should be aware of. In addition, some applications require interchangeability between HMOS and CHMOS parts. The differences that need to be considered are as follows:

**External Clock Drive:** To drive the HMOS 8051 with an external clock signal, one normally grounds the XTAL1 pin and drives the XTAL2 pin. To drive the CHMOS 8051 with an external clock signal, one must drive the XTAL1 pin and leave the XTAL2 pin unconnected. The reason for the difference is that in the

HMOS 8051, it is the XTAL2 pin that drives the internal clocking circuits, whereas in the CHMOS version it is the XTAL1 pin that drives the internal clocking circuits.

There are several ways to design an external clock drive to work with both types. For low clock frequencies (below 6MHz), the HMOS 8051 can be driven the same way as the CHMOS version, namely, through XTAL1 with XTAL2 unconnected. Another way is to drive both XTAL1 and XTAL2; that is, drive XTAL1 and use an external inverter to derive from XTAL1 a signal with which to drive XTAL2.

In either case, a 74HC or 74HCT circuit makes an excellent driver for XTAL1 and/or XTAL2, because neither the HMOS nor the CHMOS XTAL pins have TTL-like input logic levels.

**Unused Pins:** Unused pins of Ports 1, 2, and 3 can be ignored in both HMOS and CHMOS designs. The internal pullups will put them into a defined state. Unused Port 0 pins in 8051 applications can be ignored, even if they're floating. But in 80C51BH applications, these pins should not be left afloat. They can be externally pulled up or down, or they can be internally pulled down by writing 0s to them.

8031/80C31BH designs may or may not need pullups on Port 0. Pullups aren't needed for program fetches, because in bus operations the pins are actively pulled high or low by either the 8031 or the external program memory. But they are needed for the CHMOS part if the Idle or Power Down mode is invoked, because in these modes Port 0 floats.

**Logic Levels:** If  $V_{CC}$  is between 4.5V and 5.5V, an input signal that meets the HMOS 8051's input logic levels will also meet the CHMOS 80C51BH's input logic levels (except for XTAL1/XTAL2 and RST). For the same  $V_{CC}$  condition, the CHMOS device will reach or surpass the output logic levels of the HMOS device. The HMOS device will not necessarily reach the output logic levels of the CHMOS device. This is an important consideration if HMOS/CHMOS interchangeability must be maintained in an otherwise CMOS system.

HMOS 8051 outputs that have internal pullups (Ports 1, 2, and 3) "typically" reach 4V or more if  $I_{OH}$  is zero, but not fast enough to meet timing specs. Adding an external pullup resistor will ensure the logic level, but still not the timing, as shown in Figure 23. If timing is an issue, the best way to interface HMOS to CMOS is through a 74HCT circuit.

**Idle and Power Down:** The Idle and Power Down modes exist only on the CHMOS devices, but if one

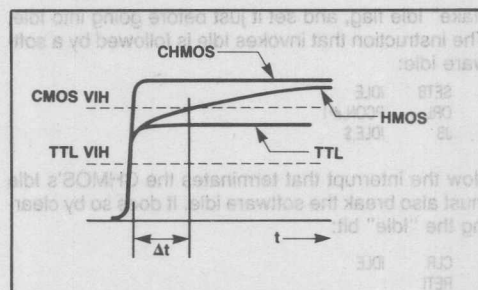


Figure 23. 0-to-1 Transition shows unspc'd delay ( $\Delta t$ ) in HMOS to 74HC Logic.

wishes to preserve the capability of interchanging HMOS and CHMOS 8051s, the software has to be designed so that the HMOS parts will respond in an acceptable manner when a CHMOS reduced power mode is invoked.

For example, an instruction that invokes Power Down can be followed by a "JMP \$":

```
CLR EA
ORL PCON,#2
JMP $
```

The CHMOS and HMOS parts will respond to this sequence of code differently. The CHMOS part, going into a normal CHMOS Power Down Mode, will stop fetching instructions until it gets a hardware reset. The HMOS part will go through the motions of executing the ORL instruction, and then fetch the JMP instruction. It will continue fetching and executing JMP \$ until hardware reset.

Maintaining HMOS/CHMOS 8051 interchangeability in response to Idle requires more planning. The HMOS part will not respond to the instruction that puts the CHMOS part into Idle, so that instruction needs to be followed by a software idle. This would be an idling loop which would be terminated by the same conditions that would terminate the CHMOS's hardware Idle. Then when the CHMOS device goes into Idle, the HMOS version executes the idling loop, until either a hardware reset or an enabled interrupt is received. Now if Idle is terminated by an interrupt, execution for the CHMOS device will proceed after RETI from the instruction following the one that invoked Idle. The instruction following the one that invoked Idle is the idling loop that was inserted for the HMOS device. At this point, both the HMOS and CHMOS devices must be able to fall through the loop to continue execution.

One way to achieve the desired effect is to define a



"fake" Idle flag, and set it just before going into Idle. The instruction that invokes Idle is followed by a software idle:

```
SETB IDLE
ORL PCON,#1
JB IDLE,$
```

Now the interrupt that terminates the CHMOS's Idle must also break the software idle. It does so by clearing the "Idle" bit:

```
CLR IDLE
RETI
```

Note too that the PCON register in the HMOS 8051 contains only one bit, SMOD, whereas the PCON register in CHMOS contains SMOD plus four other bits. Two of those other bits are general purpose flags. Maintaining HMOS/CHMOS interchangeability requires that these flags not be used.

## References

1. Pawloski, Moroyan, Altnether, "Inside CMOS

Technology," *BYTE magazine*, Sept., 1983. Available as Article Reprint AR-302.

2. Kokkonen, Pashley, "Modular Approach to C-MOS Technology Tailors Process to Application," *Electronics*, May, 1984. Available as Article Reprint AR-332.

3. Williamson, T., *Designing Microcontroller Systems for Electrically Noisy Environments*, Intel Application Note AP-125, Feb. 1982.

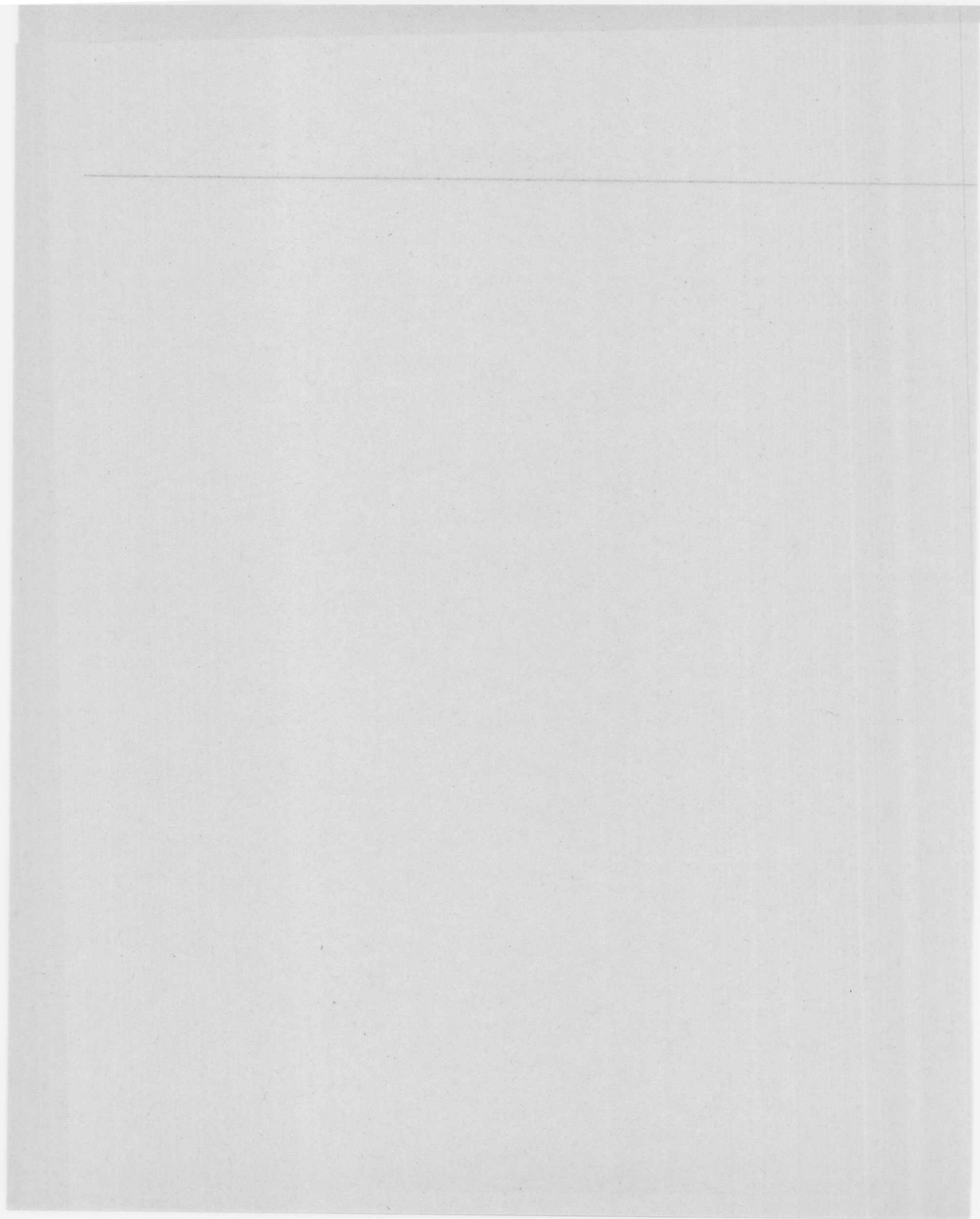
4. Williamson, T., "PC Layout Techniques for Minimizing Noise," *Mini-Micro Southeast*, Session 9, Jan. 1984.

5. Altnether, J., *High Speed Memory System Design Using 2147H*, Intel Application Note AP-74, March 1980.

6. Ott, H., "Digital Circuit Grounding and Interconnection," *Proceedings of the IEEE Symposium on Electromagnetic Compatibility*, pp. 292-297, Aug. 1981.

7. *Digital Sensors by Technar*, Technar Inc., 205 North 2nd Ave., Arcadia, CA 91006.





January 1985

## Build-In Basic Meter

## Chip Turns Into Core

**John Katausky**  
Intel Corporation  
Chandler, AZ

John Katsaros, Intel Corp.

ORDER NUMBER: 270051-001



## DESIGN ENTRY

# Built-in Basic interpreter turns controller chip into versatile system core

*An extended set of Basic statements and operators transforms a chip into the foundation for a wide range of embedded real-time systems.*

Single-chip computers have been popular for some time now, especially in embedded applications. But the phrase "computer on a chip" takes on a whole new meaning with the appearance of a software package that implements a version of Basic in silicon.

The 8052AH-Basic controller is aimed primarily at data acquisition, test instrumentation, and process control. Further, it is right at home in virtually any embedded environment and monitoring system.

Unlike so-called Tiny Basic interpreters, the chip's MCS Basic-52 software package is a full Basic interpreter. In addition, it can manipulate strings and handle logical operators (AND, OR, exclusive-OR, and NOT) and floating-point arithmetic. It is also able to accept and deliver numbers in floating-point, integer, or hexadecimal formats. Counted among its other features are built-in EPROM programming, mnemonic access to all on-chip I/O resources, and a built-in real-time clock with a resolution to within 5 ms. Moreover, its complete function library of routines can be accessed in assembly language (see the table, opposite). And the package supplements Basic's standard instructions with commands designed specifically for embedded

systems. The package resides in the 8-kbit ROM of the 8052AH-Basic chip. Apart from the software, it is identical with the 8052AH and so features 256 bytes of RAM and three multiple-mode 16-bit timer-counters.

Although Basic is held in low regard by many computer scientists, one fact cannot be denied: It is by far the most popular microcomputer language. Not only is virtually every personal or home computer capable of running it, but the language has become a de facto standard for many applications, including laboratory and

**Summary of Basic-52's features**

| Commands      | Statements      | Operators              |
|---------------|-----------------|------------------------|
| RUN           | BAUD            | INPUT                  |
| LIST          | CALL            | LET                    |
| LIST#         | CLEAR           | ONERR                  |
| NEW           | <b>CLEAR#</b>   | <b>ONEXT1</b>          |
| NULL          | <b>CLEAR1</b>   | <b>ONTIME</b>          |
| <b>RAM</b>    | <b>CLOCK0</b>   | PRINT                  |
| <b>ROM</b>    | <b>CLOCK1</b>   | <b>PRINT#</b>          |
| <b>XFER</b>   | DATA            | <b>PH0.</b>            |
| <b>PROG</b>   | READ            | <b>PH0.#</b>           |
| <b>PROG1</b>  | RESTORE         | <b>PH1.</b>            |
| <b>PROG2</b>  | DIM             | <b>PH1.#</b>           |
| <b>FPROG</b>  | <b>DO-WHILE</b> | <b>PUSH</b>            |
| <b>FPROG1</b> | <b>DO-UNTIL</b> | <b>POP</b>             |
| <b>FPROG2</b> | END             | <b>PWM</b>             |
|               | FOR-TO-STEP     | REM                    |
|               | NEXT            | <b>RETI</b>            |
|               | GOSUB           | STOP                   |
|               | RETURN          | <b>STRING</b>          |
|               | GOTO            | U0                     |
|               | ON-GOTO         | <b>U1</b>              |
|               | ON-GOSUB        | <b>U00</b>             |
|               | IF-THEN-ELSE    | <b>U01</b>             |
|               |                 | PI                     |
|               |                 | ASC ( )                |
|               |                 | CHR ( )                |
|               |                 | EXPONENTIATION ( ** )  |
|               |                 | MULTIPLY ( * )         |
|               |                 | SUBTRACT ( - )         |
|               |                 | LOGICAL AND ( .AND. )  |
|               |                 | LOGICAL OR ( .OR. )    |
|               |                 | LOGICAL X-OR ( .XOR. ) |
|               |                 | LOGICAL NOT            |
|               |                 | ABS ( )                |
|               |                 | INT ( )                |
|               |                 | SGN ( )                |
|               |                 | SQR ( )                |
|               |                 | RND                    |
|               |                 | LOG ( )                |
|               |                 | EXP ( )                |
|               |                 | SIN ( )                |
|               |                 | COS ( )                |
|               |                 | TAN ( )                |
|               |                 | ATN ( )                |
|               |                 | =, >, <, =, <          |
|               |                 | PORT1                  |
|               |                 | PCON                   |
|               |                 | RCAP2                  |
|               |                 | T2CON                  |
|               |                 | TCON                   |
|               |                 | TMOD                   |
|               |                 | TIME                   |
|               |                 | TIMER0                 |
|               |                 | TIMER1                 |
|               |                 | TIMER2                 |
|               |                 | TIME                   |
|               |                 | XTAL                   |
|               |                 | MTOP                   |
|               |                 | LEN                    |
|               |                 | FREE                   |

Boldface statements have been added specifically for embedded systems.

**John Katausky, Intel Corp.**

John Katausky is technical marketing manager for Intel Corp.'s microcontroller operation in Chandler, Ariz.

office automation. Its popularity is easily explained by its simplicity and ease of use. A programmer can execute, edit, and debug short programs in minutes, without having to worry about compilation, linking, and locating.

### Building up Basic

Like any language, Basic has its drawbacks, but the 8052AH-Basic's software package addresses many of them by beefing up its instruction set. These enhancements address three general areas: First, sophisticated programmers dislike the fact that Basic is an unstructured language. Because GO TO statements can be placed anywhere in a program, understanding and debugging it can become quite a chore. For that reason, the package contains the more structured DO-WHILE and DO-UNTIL statements, as well as the standard FOR-NEXT and IF-THEN-ELSE. These extensions furnish better ways to control the flow of a program.

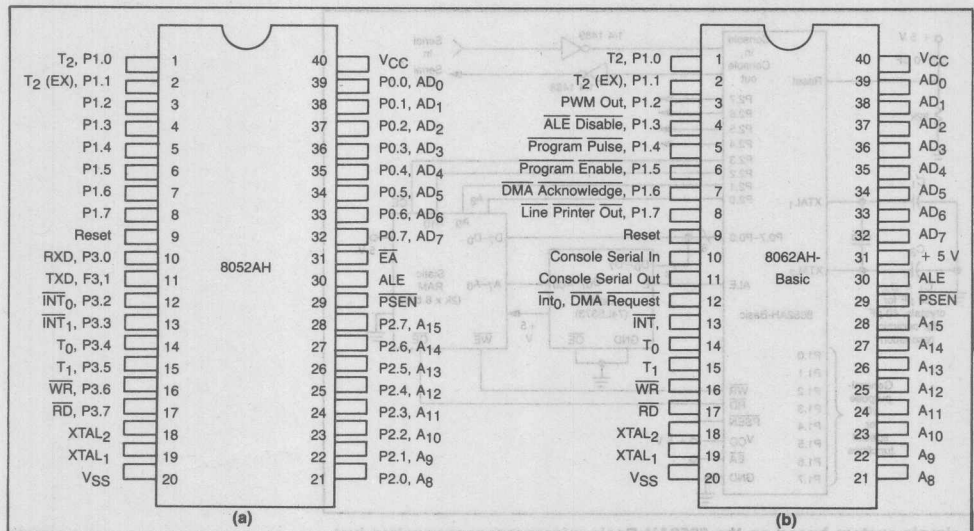
Second, the fact that Basic uses global variables forces subroutines to use the same names for variables as the main program employs. Generally, this is an inconvenience because

variables may need to be redefined every time a subroutine is called up. The package's function library, however, includes PUSH and POP statements that allow the user to pass variables to subroutines on a stack, making it easy to re-define variables in a subroutine. Additionally, PUSH and POP can be used to pass parameters to and from user-supplied assembly language routines.

Finally, interpreted Basic is usually a slow language. Although this is a legitimate complaint, many applications can still be adequately served by it. The package's fast token-based interpreter, though, puts an end to the problem, turning in execution speeds that compare quite favorably with those of popular 8- and 16-bit personal computers.

### A quick turn around the pins

As mentioned, the software is stored in an 8052AH controller chip, which is itself housed in a standard 40-pin DIP (Fig. 1). Some of the pins' uses are predetermined. Of the four ports on the standard 8052AH, only port 1 still serves as an I/O port on the Basic version. Since the



1. The pin assignments of the 8052AH (a) and the 8052AH-Basic controller (b) differ primarily in that the second does not use lines AD<sub>0</sub> to AD<sub>7</sub> and A<sub>8</sub> to A<sub>15</sub> as ports 0 and 2, respectively. Further, the Basic version needs +5 V on pin 31 to enable the 8052AH to execute from internal ROM.

## DESIGN ENTRY

8052AH-Basic requires external memory, Pins 32 through 39 and 21 through 28, which formerly served ports 0 and 2 respectively, are now used for their alternative function: addressing and exchanging data with RAM and ROM. Pins 10 through 17, which formed port 3, are putting in their time at their assorted alternative duties—with some slight revisions. The only other changes involve pin 31, EA, which is now tied to  $V_{CC}$ , and port 1, whose pins have acquired a second set of functions.

### Port 1 in detail

To start with, pins 1 and 2 of port 1 (P1.0 and P1.1) can also be used to clock and trigger timer-counter 2. Pins 3, 4, and 5 generate all timing and other signals necessary to program just about any EPROM or EEPROM—a simple way to save and retrieve programs without uploading or downloading. Pin 6, together with the  $INT_0$  pin (pin 12), gives the user the option of implementing direct memory access. Finally, pin 7 takes care of direct serial output to a peripheral, such as a printer. All that needs to be done in this case is to invoke the LIST# com-

mand or the PRINT# statement.

In addition, the syntax of Basic-52 permits the user to read and write directly from and to port 1. For instance, the statement:

```
PORT1 = 55H
```

would place the hexadecimal value 55 on the appropriate pins, while:

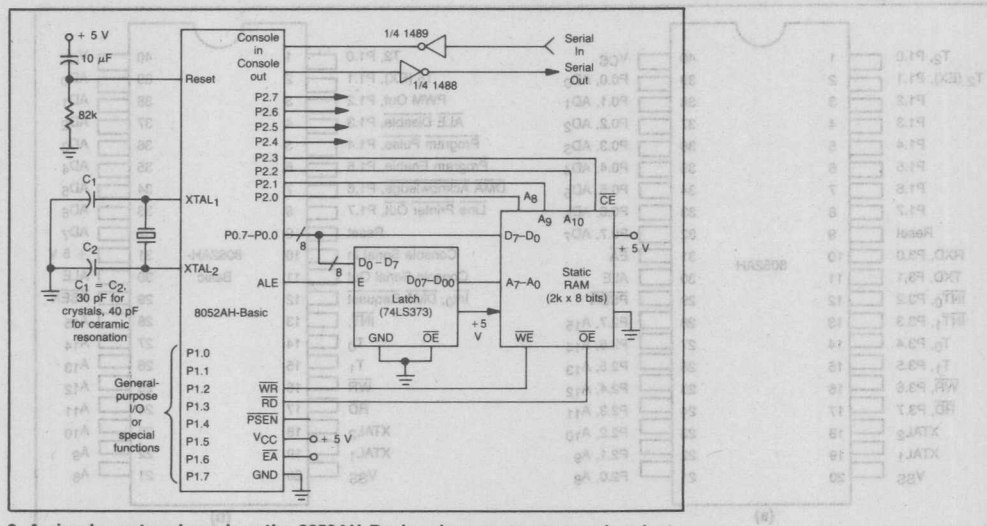
```
A = PORT1
```

would assign the value of variable A to the port 1 pins. Furthermore, some of the language's commands and statements manipulate those pins so that they furnish a number of useful and unique features. For example, P1.2 can also serve as an output for the pulse-width modulate (PWM) statement. Executing it generates a pulse train of varying duration, frequency, and duty cycle on the pin.

The statement:

```
PWM 50, 100, 1000
```

creates 1000 cycles of a signal that, assuming a 12-MHz clock, is high for 50  $\mu$ s and low for 100  $\mu$ s. The PWM statement also can generate audi-



2. A simple system based on the 8052AH-Basic microprocessor requires just three additional chips plus the serial-port driver. Port 1 could be used, for example, to sense switches, drive indicators, or a power printer.

ble feedback in process control and security systems.

To pin 9 (Reset) falls the job of initializing the system. It also furnishes power on reset when an external 8.2-k $\Omega$  resistor is connected from this pin to ground and a 10- $\mu$ F capacitor is connected to  $V_{CC}$ .

Under the software package, all of the pins that formed port 3 are pressed into service in their alternative functions. One of the interrupt pins,  $INT_0$ , now also handles DMA requests (pin 12), and the pins for transmitting and receiving data (pins 11 and 12) now handle serial inputs and outputs to the console. Their activities are integrated into a Basic application program. For example, an interrupt on the  $INT_1$  pin (pin 13) can be handled by the ONEXT1 statement:

```
10 ONEXT1 1000
```

which forces a GOSUB to line 1000 every time the  $INT_1$  pin is pulled to a logical 0. In the called subroutine, the user can process the interrupt, then use a RETI statement to exit from the interrupt handler.

To turn the Basic chip into a complete system, the user need supply only external RAM and serial port drivers (Fig. 2). The RAM requirements are few: At least 1 kbyte must be present, and it must start at external memory location 0 and be contiguous and completely decoded. After reset, the package determines how much memory is present in the system and initializes it, then waits for the user to type in a space character. That character establishes the baud rate automatically.

Although this system is quite useful, it does not represent an ideal embedded controller. Fortunately, adding just a little more hardware is all that is needed to construct one complete with automatic self-starting, EPROM and EEPROM programming, and simple program-file management (see "EPROM Saves the Program," opposite).

Once the desired type of EPROM is in place, starting up the system involves no more than entering either the PROG2 or FPROG2 command. Both commands preserve the baud rate information in exactly the same way as PROG1 or FPROG1 does. In addition, though, either causes the first program stored in PROM to ex-

## EPROM saves the program

EPROM and EEPROM programming is certainly one of the most powerful features of the MCS Basic-52 software package. And it is simple to use; only three of the 8052AH-Basic microprocessor's control pins are needed to call it into play. In brief, after the user enters a program, it is stored in RAM, starting at location 512. When the user types PROG or FPROG, the program that is stored in RAM is programmed into an EPROM or EEPROM whose address is 8000<sub>16</sub>.

To start the process, the CPU first writes a logical 0 to the Program Enable pin, P1.5, thus supplying the higher voltage required to program the EPROM. Next, the processor reads the appropriate RAM location, starting at 512, and saves the byte of information.

Then, the CPU sends out the appropriate low-order PROM address to pins  $AD_0$  to  $AD_7$  and sets the ALE Disable pin to 0. That latches the low-order PROM address permanently into the system address latch—74LS373 (see the figure). Only one AND gate is required to do this. The processor then writes the high-order PROM address to pins  $A_8$  through  $A_{15}$  and the data to pins  $AD_0$  to  $AD_7$ . Finally, a 0 is written to the Program Pulse pin. Depending on the PROM used, that pin is held low for either 1 or 50 ms. After it returns to a logical 1, the CPU verifies the contents of the programmed PROM and then writes a 1 to ALE Disable.

The process continues until the entire Basic program is saved in PROM. When programming is complete, the CPU writes a 1 to the Program Enable pin, leaves the programming routine, and returns to the Basic command mode. To prevent accidental programming during power-up, OR gates should be used (top left in figure).

The foregoing description of the programming naturally leads to two questions.

The first concerns the duration of the 0 state. A 50-ms pulse is required to program most standard EPROMs. The 1-ms pulse, on the other hand, permits the software package to program Intel's latest generation of high-density EPROMs, working with the company's programming algorithm, dubbed INTELigent. To apply it, the user simply types FPROG instead of PROG. The algorithm requires that the  $V_{CC}$  on the EPROM be increased to 6 V. One economical way to accomplish this is with a DIP relay. The software package also allows the user to specify programming pulses of any length.

The other question centers on the accuracy of the programming pulse versus that of the system clock frequency. To eliminate any dependency on the



XTAL = 12000000

a 12-MHz system clock is assumed and the proper PROM programming time calculated. The real-time clock pulses and the baud rate for the line printer output port are also determined. This ability gives the designer the freedom to choose the desired system clock frequency without having to worry about the internal timing of the Basic software.

with the file number that has been assigned to that program. To call up a given program, the user simply types ROM X, and the package finds the appropriate one. Typing RUN starts program execution.

Suppose the user discovers an error in a program that is stored in EPROM—normally bad news because such programs cannot be edited. However, the package allows a program to be transferred from EPROM to RAM by typing XFER. Once XFER is entered, the program may be edited like any other. Typing PROG again saves the edited program.

Additionally, after reset, the software package waits for a space character, from which the baud rate is derived, to be entered on the serial port. As an option, the package can be made, on receiving the command PROG1 or FPROG1, to save the serial-port baud rate information in PROM. The next time the processor is reset, the package can sign on directly without the need to enter a space character.



ecute directly after reset, bypassing a RUN command. In fact, even the console is no longer needed in some embedded systems. Programs can be written and debugged with the aid of a terminal, and when the programmer is satisfied with the results, the hardware can be embedded in the design and the terminal disconnected.

#### On call

Aside from the features already noted, the package contains a host of others that are very attractive for embedded systems. The CALL statement, for instance, accesses assembly-language routines directly from the package, a handy way in which to meet speed requirements. Better still, assembly-language programs can take advantage of a complete library of routines that reside in the package. The user gains access to this library simply by placing a specific op code in the accumulator and then sending a CALL to location 30<sub>16</sub>.

Suppose, for example, that the assembly-language routine reads a 16-bit value, and the user would like to calculate the value's square root and send it to a particular location. Assuming that the user supplies the 16-bit integer to the high- and low-byte registers, R<sub>2</sub> and R<sub>0</sub>,

respectively, the calculation can be made simply (Program 1). All told, over 60 routines can be invoked by assembly-language programs.

The software package also controls all of the system's I/O memory resources. To make things simple, all of these operations are symmetrical. For example, the operator DBY in the statement:

A = DBY (0FEH)

assigns the value located at 0FE<sub>16</sub> in the chip's internal memory to variable A. The statement:

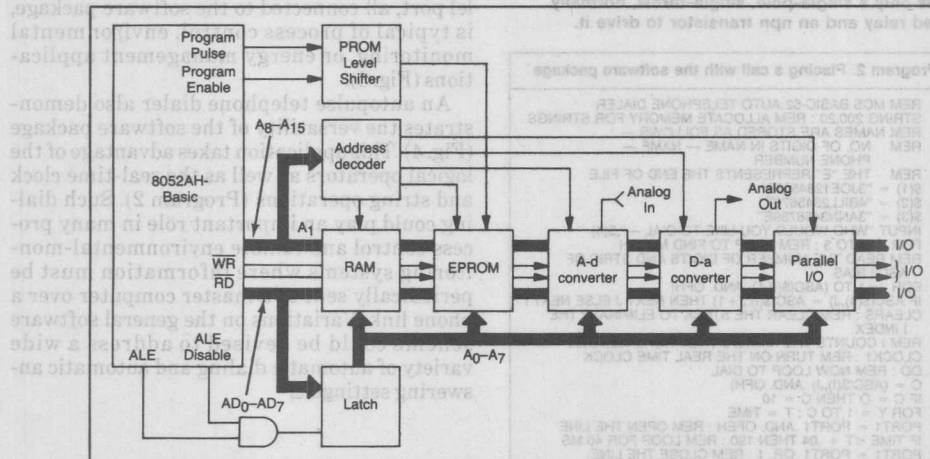
DBY (0FEH) = 22H

places 22<sub>16</sub> in internal memory location 0FE<sub>16</sub>. The same symmetry holds for the interrupt registers, IE and IP, as well as the timers and the timer configuration registers.

Other special instructions are particularly

#### Program 1. Getting to the root of a variable

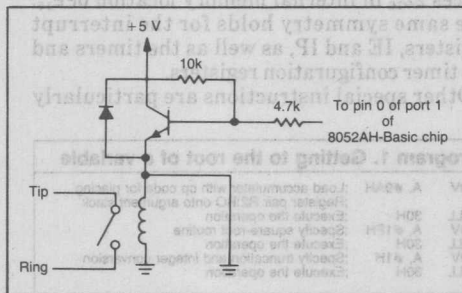
|      |         |  |
|------|---------|--|
| MOV  | A, #9AH | :Load accumulator with op code for placing |
| CALL | 30H     | :Register pair R2:R0 onto argument stack   |
| MOV  | A, #1FH | :Execute the operation                     |
| CALL | 30H     | :Specify square-root routine               |
| MOV  | A, #1H  | :Execute the operation                     |
| CALL | 30H     | :Specify truncation and integer conversion |
| CALL | 30H     | :Execute the operation                     |



3. With the help of an address decoder, the I/O on a system built around the Basic chip can be expanded at will. One such setup converts analog into digital and digital into analog signals, and supplies a parallel I/O port for a console.

# DESIGN ENTRY

helpful for the designer. Thus BAUD sets the baud rate for the line printer port. If the designer wants to transmit at 1200 bauds, he or she simply enters BAUD 1200. CLEARS is used to reset all of the package's stacks, and CLEARLI resets all interrupts, except the real-time clock. CLOCK1 enables the real-time clock, just as CLOCK0 disables it. The value of the clock can be read or assigned by using the TIME operator. ONTIME generates an interrupt when the TIME operator reaches or exceeds a specific value. ONTIME works in much the same way as the aforementioned ONEXT1 statement. Finally, the statements PH0 and PH1, when used to-



4. To function as an autopulse dialer, the Basic chip needs only a single-pole, single-throw, normally closed relay and an npn transistor to drive it.

## Program 2. Placing a call with the software package

```

10 REM MCS BASIC-52 AUTO TELEPHONE DIALER
20 STRING 200,20 : REM ALLOCATE MEMORY FOR STRINGS
30 REM NAMES ARE STORED AS FOLLOWS —
40 REM NO. OF DIGITS IN NAME — NAME —
   PHONE NUMBER
50 REM THE "E" REPRESENTS THE END OF FILE
60 $(1) = "3JOE1234567E"
40 $(2) = "4BILL2345678E"
50 $(3) = "3ANN3456789E"
60 INPUT "WHO WOULD YOU LIKE TO DIAL —":$(4)
70 FOR I = 1 TO 3 : REM LOOP TO FIND MATCH
80 REM READ THE NUMBER OF DIGITS AND STRIP OF
   ASCII BIAS
90 FOR J = 1 TO (ASC$(I),1) AND (OFH)
100 IF ASC$(4),J = ASC$(I),J+1 THEN NEXT J ELSE NEXT I
110 CLEARS : REM CLEAR THE STACK TO ELIMINATE THE
   INDEX
120 REM I COUNTS THE NAME, J THE PHONE NUMBER
130 CLOCK1 : REM TURN ON THE REAL TIME CLOCK
140 DO : REM NOW LOOP TO DIAL
150 C = (ASC$(I),J) AND (OFH)
160 IF C = 0 THEN C = 10
170 FOR Y = 1 TO C : T = TIME
180 PORT1 = PORT1 AND (OFH) : REM OPEN THE LINE
190 IF TIME < T + .04 THEN 190 : REM LOOP FOR 40 MS
200 PORT1 = PORT1 OR 1 : REM CLOSE THE LINE
210 IF TIME < T + .1 THEN 210 : REM LOOP FOR 60 MS
220 NEXT Y
230 J = J + 1 : WHILE ASC$(I),J <> ASC(E)
240 END

```

gether, ensure that outputs are given in hexadecimal notation. PH0 suppresses leading zeros, and PH1 always prints out four hexadecimal digits.

Building a functioning system around the Basic chip is a snap. The designer can link to virtually any standard peripheral chip simply by assigning some location to the peripheral in the 64 kbytes of external address available to the 8-bit device.

## Addressing I/O

In fact, the software actually reserves the chip's upper 8 kbytes of memory (from 56 to 64 kbytes) for I/O. Accessing external peripherals is accomplished by using the XBY, or address, operator. For example:

A = XBY (OFFF0H)

reads a peripheral or external memory location that has been decoded for address OFFF0<sub>16</sub> and assigns its value to the variable A. Similarly:

XBY (OFFF0H) = 55H

writes 55<sub>16</sub> to a peripheral or memory location that has been decoded for address OFFF0<sub>16</sub>. A system consisting of an analog-to-digital and a digital-to-analog converter plus an 8255 parallel port, all connected to the software package, is typical of process control, environmental monitoring, or energy management applications (Fig. 3).

An autopulse telephone dialer also demonstrates the versatility of the software package (Fig. 4). This application takes advantage of the logical operators as well as the real-time clock and string operations (Program 2). Such dialing could play an important role in many process control and remote environmental-monitoring systems where information must be periodically sent to a master computer over a phone link. Variations on the general software scheme could be devised to address a wide variety of automatic dialing and automatic answering settings. □

# INCREASED FUNCTIONS IN CHIP RESULT IN LIGHTER, LESS COSTLY PORTABLE COMPUTER

October 1985

Jatet Modares, Applications Engineer, Intel Corp., Chandler, AZ

displaying the same number of characters as a typical CRT, and it is not as bulky or heavy as the latter.

LCDs can be divided into two large categories: smart LCDs and dumb LCDs. Those displays that have the capability to receive a byte of ASCII and translate it into a displayed character are considered smart. On the other hand, dumb displays use only a matrix of dots. The former type has some kind of controller of its own plus a small memory to hold the look-up table of characters (ROM or EPROM). When using a microcontroller, the microcontroller has to manage the correct data to be displayed. This means more ROM and a more complex design. However, it gives the user special or custom character sets of displays normally accessed through an 8-bit bus. Although the LCD is relatively slow, it can still share the bus with a memory device without any degradation of the system performance.

**Keyboards**  
Depending on their task and purpose, keyboards vary in size, and the number of keys. The keyboard for a computer terminal is a minimum; should have all the alphanumeric keys (standard typewriter) plus a Control, an Escape, and optionally, some function

communicates with the host computer at very high BAUD rates, and displays information on the screen at lower rates for human beings. The chip also monitors power supply for switching to the case of power failure to save valuable computer time. The microcontroller can be interfaced to a host computer through a parallel or serial interface. It can be connected to a host computer through a parallel or serial interface. It can be connected to a host computer through a parallel or serial interface.

The display device of this portable terminal is a 256-character 1-line monochrome Display (LCD). It is capable of displaying 256 characters in a single line. It is capable of displaying 256 characters in a single line. It is capable of displaying 256 characters in a single line.

The display device of this portable terminal is a 256-character 1-line monochrome Display (LCD). It is capable of displaying 256 characters in a single line. It is capable of displaying 256 characters in a single line. It is capable of displaying 256 characters in a single line.

Advances in technology have made it possible to reduce the size and increase the functionality and performance of computers and computer peripherals. With the help of microtechnology it is possible to construct a computer terminal that is smaller and lighter than a briefcase, and that can be connected to a host computer through a parallel or serial interface. It can be connected to a host computer through a parallel or serial interface.

As more portable computers are introduced to the marketplace, the demand for lighter and even smaller systems at a lower cost are inevitable. To meet this demand changes in the architecture are necessary.

With Intel's recently introduced 80C21BH microcontroller several options in the design of the portable computer have been overcome. The 80C21BH is a single chip 8-bit microcontroller that requires a single 5V power supply. It has 32 I/O lines. Its functionalities include excellent bit and byte manipulation capability at extremely high speeds as well as interfacing facilities through the serial and parallel channels to intelligent and unintelligent devices. It can carry its own program memory up to 4 Kbytes and has various tools and support systems.

This article discusses the implementation of a prototype portable terminal based on Intel's new 80C21BH microcontroller, and introduces the tools and techniques available to build such a computer terminal.

© INTEL CORPORATION, 1985

ORDER NUMBER: 270126-001

Reprinted from Design News, 8-19-85



# INCREASED FUNCTIONS IN CHIP RESULT IN LIGHTER, LESS COSTLY PORTABLE COMPUTER

October 1988

Jafar Modares, Applications Engineer, Intel Corp., Chandler, AZ

Advances in technology have made it possible to reduce the size and increase the functionality and performance of computers and computer peripherals. With the help of microtechnology it is possible to construct a computer terminal that is smaller and lighter than a briefcase, and that can be connected to a mainframe from almost anywhere.

As more portable computers are introduced to the marketplace, the demand for lighter and even smaller systems at a lower cost are inevitable. To meet this demand changes in the architecture are necessary.

With Intel's recently introduced 80C51BH microcontroller several obstacles in the design of the portable computer have been overcome. The 80C51BH is a single chip 8-bit microcontroller that requires a single 5V power supply. It has 32 I/O lines. Its functionalities include excellent bit and byte manipulation capability at extremely high speeds as well as interfacing flexibilities through the serial and parallel channels to intelligent and unintelligent devices. It can carry its own program memory up to 4 Kbytes and has various tools and support systems.

This article discusses the implementation of a prototype portable terminal based on Intel's new 80C51BH microcontroller, and introduces the tools and techniques available to build such a computer terminal. In the application discussed, the chip monitors the keyboard,

communicates with the host computer at very high BAUD rates, and displays information on the screen at slower rates for human beings. The chip also monitors the power supply for switching to the battery in case of power failure to save valuable data and computer time. The prototype is currently under further development at Intel's Microcontroller Division.

## Introducing the 80C51BH

Very new in the market, the 80C51BH is a member of Intel's MCS-51 family. The MCS-51 is a group of 8-bit microcontrollers that are extremely powerful because of their I/O structure and their bit manipulating capabilities. The 80C51BH has 4 Kbytes of on-chip program memory with the capability to address another 60 Kbytes of external program memory. In addition it has 128 bytes of on-chip RAM and can access 64 Kbytes of external data memory.

Since the 80C51BH is CMOS, it has very low power consumption (15 mA at 5V, 12 MHz). In addition, it has two power saving features not available in HMOS versions of the family. These are the Idle and Power Down modes, which are controlled by software and further reduce power consumption. These power saving features make it ideal for battery-operated backed-up systems.

## Display device

The display device of this portable terminal is a 25-line  $\times$  80-character Liquid Crystal Display (LCD). It is capable of

displaying the same number of characters as a typical CRT, and it is not as bulky or heavy as the latter.

LCDs can be divided into two large categories: smart LCDs and dumb LCDs. Those displays that have the capability to receive a byte of ASCII and translate it into a displayed character are considered smart. On the other hand, dumb displays are only a matrix of dots. The former type has some kind of controller of its own plus a small memory to hold the look-up table of characters (character generator). When using a dumb display the microcontroller has to address and turn on the correct dots to make a character, this means more I/O pins will be required. However, it gives extra capabilities such as graphic displays and special or custom character generation.

Both types of displays normally accept data through an 8-bit bus. Although the LCD is relatively slow, it can still share the bus with a memory device without any degradation of the system performance.

## Keyboard

Depending on their task and purpose, keyboards vary in shape, size, and the number of keypads. The keyboard for a computer terminal, as a minimum, should have all the alphanumeric keys (standard typewriter) plus a Control, an Escape, and optionally, some Function keys.

As the diagram in Figure 1 shows a

typical keyboard consists of a matrix of eight scan lines and eight receive lines. The scan lines are connected to Port 0 of the microcontroller, and receive lines are connected to Port 1. The software writes 0s to Port 0 to hold the scan lines at a logic Low, and it writes 1s to Port 1 to hold the receive lines at a logic High. Pressing one of the keys connects a scan line to receive a line and pulls that receive line Low.

Besides being used for the scan lines, Port 0 is also the bus for the data buffer RAM and the display unit. While the controller is talking to the RAM or to the display, the bus must not be used by other devices or bus contention can occur. Since the microcontroller initiates the access to the display and to the data buffer RAM, no conflict can occur between them. However, if more than one key on the same receive line is held down simultaneously while the RAM or the display is being accessed, it is possible to foul up the information being transferred. Thus, to avoid bus contention, diodes D1 through D8 are placed on the scan lines, as shown in Figure 1.

All the receive lines are ANDed to External Interrupt 1, so that pulling any of the receive lines Low will generate an interrupt. The interrupt service routine, adapts the "two key lock-out" system and identifies the pressed key. This system allows only one key to be pressed simultaneously, all of them will be ignored.

On some keyboards, certain keys (such as Shift, Control or Escape) are not a part of the line matrix. These keys connect directly to a port pin on the microcontroller. They would not cause lock-out if pressed simultaneously with a matrix key, nor generate an interrupt if pressed singly. However, if they are part of the matrix, then the software has to recognize those keys and take proper action depending on the function of the pressed key.

Normally, when a key is pressed, the microcontroller is in the Idle mode and no other task is in progress. Pressing a key on the matrix generates an interrupt, which terminates the Idle mode. The interrupt service routine first calls a subroutine to provide a delay of approxi-

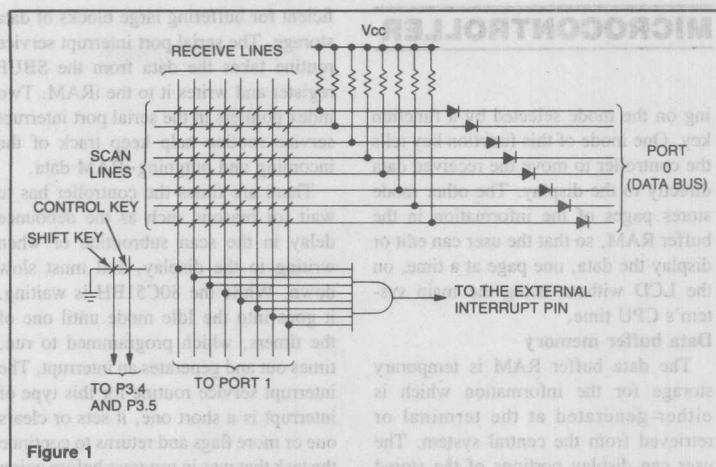


Figure 1

mately 25 msec (to debounce the key), and to perform the task of identifying which key is down.

There are a number of ways that the interrupt service routine can identify the press key. One way is to utilize the bit-addressing capabilities of the 80C51BH to test each bit of the receive lines for a zero, which generated the interrupt, and records the bit position. Then write 0s to the receive lines, 1s to the scan lines, test the scan lines for a zero, and record its position. If the controller finds more than one zero on each port, it will discontinue any further processing and return to the main program.

Once the bit positions that contain 0 are recorded, they are used as the address for the characters in the look-up table. The subroutine finds the character that corresponds to the pressed key. The character is represented in ASCII code.

The look-up table is a list of the ASCII representation of each keypad character, and is stored in the program memory. The order in which they sit corresponds to the address generated by the scan subroutine when that character is pressed on the keyboard.

#### Serial communication

Once the 80C51BH has the ASCII code generated from a key closure, it can send it through its Serial Channel to the host computer. The serial port of the microcontroller can be programmed for all of the standard rates up to 375K. If

the terminal is to be connected directly to the mainframe, a simple circuit translates the TTL levels to the RS232 level. The circuit also eliminates the need for the -12V supply required for RD232. The circuit diagram is shown in Figure 2.

However, the primary application of this terminal is to be the traveling person's window to the central system from any remote location. In this application a modem is needed. There are many types of modems available. Some are on PC boards for OEM use and others are ready to connect directly to the telephone by the user. They also vary in size, performance, and the way they communicate. A proper modem for the portable terminal should be small enough to carry in a briefcase and preferably be a low-power device. When an ASCII code is received by the host computer, it records that code and echoes it back to the microcontroller which displays it on the LCD.

The serial port of the 80C51BH can generate interrupts every time it receives or transmits information. The serial port interrupt must have service priority over the external interrupt generated by the keyboard. The high priority enables the serial port to receive data any time the computer addresses the terminal and transmits data.

The serial port interrupt service routine transfers the received data to the display or to a memory buffer, depend-

## MICROCONTROLLER

ing on the mode selected by a function key. One mode of this function key tells the controller to move the received data directly to the display. The other mode stores pages of the information in the buffer RAM, so that the user can edit or display the data, one page at a time, on the LCD without using the main system's CPU time.

### Data buffer memory

The data buffer RAM is temporary storage for the information which is either generated at the terminal or retrieved from the central system. The user can display portions of the stored information or scroll through it. The user can also alter the data on the terminal and transmit it back to the computer in a block form.

A suitable device for this purpose is the 51C86 iRAM. This device is a pseudo-static dynamic RAM that has a built-in address latch. An internal high-speed arbitration circuit resolves any potential conflict arising between read/write and internal refresh cycles.

This iRAM is  $8K \times 8$ -bit and is suf-

ficient for buffering large blocks of data storage. The serial port interrupt service routine takes the data from the SBUF register and writes it to the iRAM. Two index pointers in the serial port interrupt service routine help keep track of the incoming and outgoing iRAM data.

There are times the controller has to wait for reasons such as the debounce delay in the scan subroutine or when writing to the display, and must slow down. While the 80C51BH is waiting, it goes into the Idle mode until one of the timers, which programmed to run, times out and generates an interrupt. The interrupt service routine for this type of interrupt is a short one, it sets or clears one or more flags and returns to continue the task that was in progress before going into the Idle mode.

The controller is also in the Idle mode when there is no activity in the terminal, i.e., no data is being received or transmitted, and the keyboard is not being used. Therefore, it is appropriate to say that when power is applied to the terminal, the controller spends more than 90% of its time in the Idle mode.

### What is the Idle mode?

When Bit 0 in the Special Function Register PCON is set, the CPU gates off

its own internal clock and goes to sleep. Since the CPU consumes about 90% of the chip's power, this process saves a significant amount of power. More importantly the on chip peripherals and the RAM continue their normal functions independently of the CPU. Since the oscillator is still running, any enabled interrupt (internal or external) will wake the CPU up from its sleep, and it will start executing instructions from the interrupt service routine.

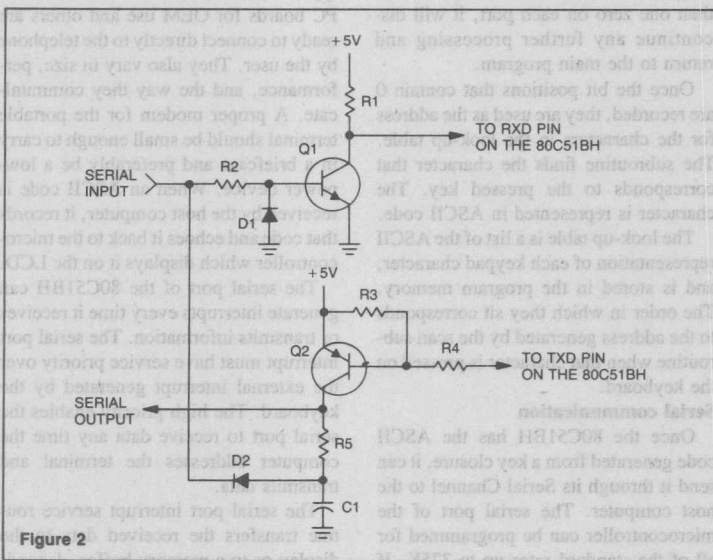
A true portable terminal should be capable of operating from a battery source. There are occasions when one would like to use the terminal at a location where an electric outlet is not readily available. But the main purpose of the battery supply is to save the data, which has been entered and stored in the buffer RAM, in the case of an unexpected power failure.

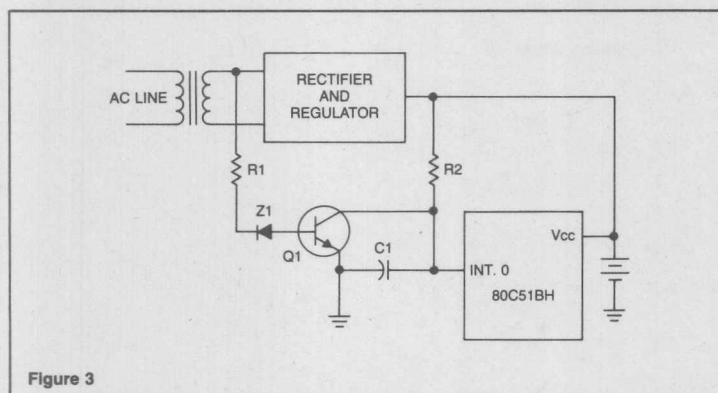
While performing all of the previously mentioned tasks, the 80C51BH can monitor its power supply, detect a power loss in its earliest stages, and initiate a power Down to save the data of the internal and external RAM.

One method for the 80C51BH to monitor its power supply is to have the positive half-cycles of the power supply transformer fed to the External Interrupt 0 pin (Figure 3). In the level activated mode, this pin generates an interrupt every time there is a high to low transition. The interrupt service routine reloads Timer 0 to a value that will make it overflow sometime between one and two periods of the line frequency. As long as the half cycles keep coming in, the timer never overflows, because it is reloaded every half a cycle. If there is a single half a cycle in which the line voltage does not reach a high enough level to generate the interrupt, the timer rolls over and generates a timer interrupt.

The interrupt service routine for Timer 0 saves the critical data of some of the internal registers and puts the controller into a Power down mode. A reset button restarts the microcontroller to resume operation when power is restored.

In this mode the CPU and all of the on-chip peripherals go to sleep. The device stops its oscillator, freezes all the





activities and saves the information in its internal RAM for as long as the supply voltage can be reduced to as low as 2 volts without running any risks of losing the internal RAM information. Supply

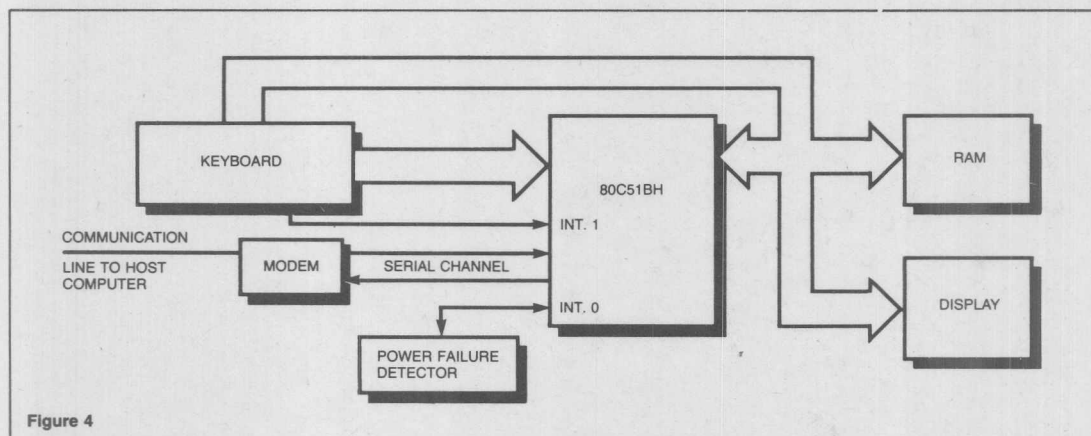
current in this mode is normally 10 to 50  $\mu$ A.

Bit 1 of the Special Function Register PCON controls this mode. The instruction that puts the device in the Power

Down mode is the last one executed. The only way for the part to exit this mode is with a hardware reset.

A prototype of this terminal was built and connected to a MDS 800 development system. The terminal communicated with the host computer through the serial channel at the rate of 2400 baud. The 80C51BH was emulated using Intel's ICE-51 in circuit emulator. The block diagram of the terminal is shown in Figure 4.

The CHMOS controller and LCD combination provides a system that requires only a single voltage power supply, and consumes less than 200 mW. The system's low power consumption eliminates the need for complex, voltage-regulating hardware, and cooling fans. The result is a lighter and smaller computer terminal at a very low cost. □







---

The Single Component  
MCS<sup>®</sup>-48 System

---

12

W.C. - 10 1/2

# CHAPTER 12

## THE SINGLE COMPONENT MCS<sup>®</sup>-48 SYSTEM

### 12.0 INTRODUCTION

Sections 12.1 through 12.4 describe in detail the functional characteristics of the 8748H and 8749H EPROM, 8048AH/8049AH/8050AH ROM, and 8035AHL/8039AHL/8040-AHL CPU only single component microcomputers. Unless otherwise noted, details within these sections apply to all versions. This chapter is limited to those functions useful in single-chip implementations of the MCS<sup>®</sup>-48. Chapter 14 discusses functions which allow expansion of program memory, data memory, and input output capability.

### 12.1 ARCHITECTURE

The following sections break the MCS-48 Family into functional blocks and describe each in detail. The following description will use the 8048AH as the representative product for the family. See Figure 14.1.

#### 12.1.1 Arithmetic Section

The arithmetic section of the processor contains the basic data manipulation functions of the 8048AH and can be divided into the following blocks:

- Arithmetic Logic Unit (ALU)
- Accumulator
- Carry Flag
- Instruction Decoder

In a typical operation data stored in the accumulator is combined in the ALU with data from another source on the internal bus (such as a register or I/o port) and the result is stored in the accumulator or another register.

The following is more detailed description of the function of each block.

#### INSTRUCTION DECODER

The operation code (op code) portion of each program instruction is stored in the Instruction Decoder and converted to outputs which control the function of each of the blocks of the Arithmetic Section. These lines control the source of data and the destination register as well as the function performed in the ALU.

#### ARITHMETIC LOGIC UNIT

The ALU accepts 8-bit data words from one or two sources and generates an 8-bit result under control of the Instruction Decoder. The ALU can perform the following functions:

- Add With or Without Carry
- AND, OR, Exclusive OR
- Increment/Decrement
- Bit Complement
- Rotate Left, Right
- Swap Nibbles
- BCD Decimal Adjust

If the operation performed by the ALU results in a value represented by more than 8-bits (overflow of most significant bit), a Carry Flag is set in the Program Status Word.

#### ACCUMULATOR

The accumulator is the single most important data register in the processor, being one of the sources of input to the ALU and often the destination of the result of operations performed in the ALU. Data to and from I/O ports and memory also normally passes through the accumulator.

#### 12.1.2 Program Memory

Resident program memory consists of 1024, 2048, or 4096 words eight bits wide which are addressed by the program counter. In the 8748H and the 8749H this memory is user programmable and erasable EPROM; in the 8048AH/8049AH/8050AH the memory is ROM which is mask programmable at the factory. The 8035AHL/8039AHL/8040AHL has no internal program memory and is used with external memory devices. Program code is completely interchangeable among the various versions. To access the upper 2K of program memory in the 8050AH, and other MCS-48 devices, a select memory bank and a JUMP or CALL instruction must be executed to cross the 2K boundary.

There are three locations in Program Memory of special importance as shown in Figure 12.2.

##### LOCATION 0

Activating the Reset line of the processor causes the first instruction to be fetched from location 0.

##### LOCATION 3

Activating the Interrupt input line of the processor (if interrupt is enabled) causes a jump to subroutine at location 3.

##### LOCATION 7

A timer/counter interrupt resulting from timer counter overflow (if enabled) causes a jump to subroutine at location 7.

Therefore, the first instruction to be executed after initialization is stored in location 0, the first word of an external interrupt service subroutine is stored in location 3, and the first word of a timer/counter service routines



# SINGLE COMPONENT MCS-48 SYSTEM

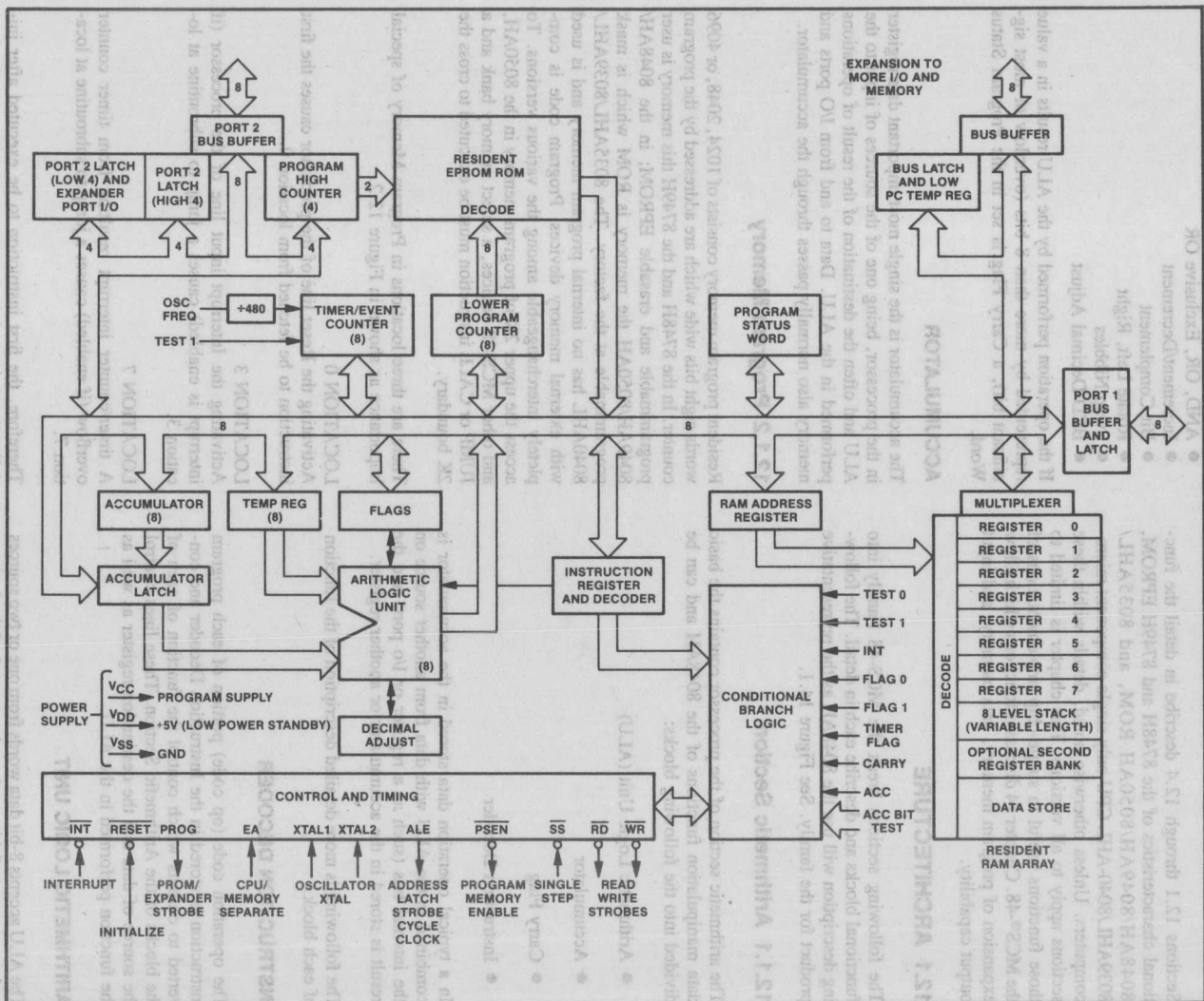


Figure 12-1. 8748H/8048AH/8749H/8049AH/8050AH Block Diagram

is stored in location 7. Program memory can be used to store constants as well as program instructions. Instructions such as MOVP and MOVP3 allow easy access to data "lookup" tables.

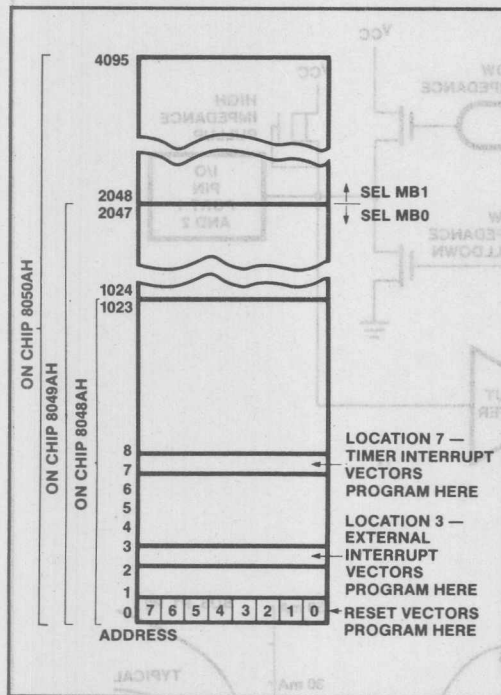


Figure 12-2. Program Memory Map

### 12.1.3 Data Memory

Resident data memory is organized as 64, 128, or 256 by 8-bits wide in the 8048AH, 8049AH and 8050AH. All locations are indirectly addressable through either of two RAM Pointer Registers which reside at address 0 and 1 of the register array. In addition, as shown in Figure 12-3, the first 8 locations (0-7) of the array are designated as working registers and are directly addressable by several instructions. Since these registers are more easily addressed, they are usually used to store frequently accessed intermediate results. The DJNZ instruction makes very efficient use of the working registers as program loop counters by allowing the programmer to decrement and test the register in a single instruction.

By executing a Register Bank Switch instruction (SEL RB) RAM locations 24-31 are designated as the working

registers in place of locations 0-7 and are then directly addressable. This second bank of working registers may be used as an extension of the first bank or reserved for use during interrupt service subroutines allowing the registers of Bank 0 used in the main program to be instantly "saved" by a Bank Switch. Note that if this second bank is not used, locations 24-31 are still addressable as general purpose RAM. Since the two RAM pointer Registers R0 and R1 are a part of the working register array, bank switching effectively creates two more pointer registers (R0' and R1') which can be used with R0 and R1 to easily access up to four separate working areas in RAM at one time. RAM locations (8-23) also serve a dual role in that they contain the program counter stack as explained in Section 12.1.6. These locations are addressed by the Stack Pointer during subroutine calls as well as by RAM Pointer Registers R0 and R1. If the level of subroutine nesting is less than 8, all stack registers are not required and can be used as general purpose RAM locations. Each level of subroutine nesting not used provides the user with two additional RAM locations.

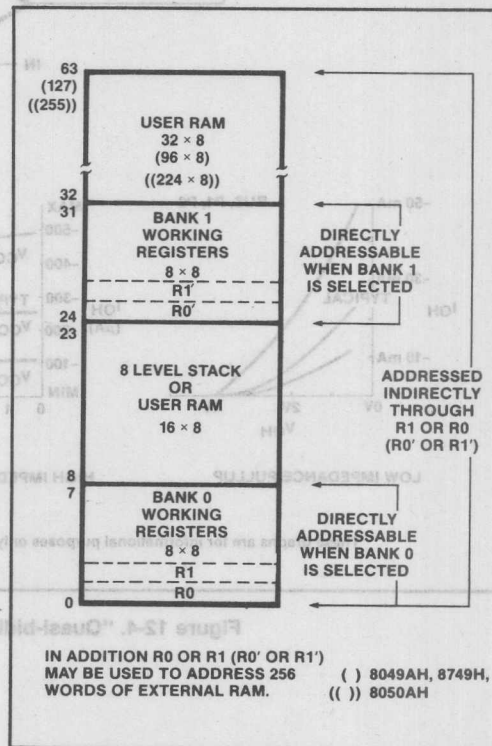


Figure 12-3. Data Memory Map

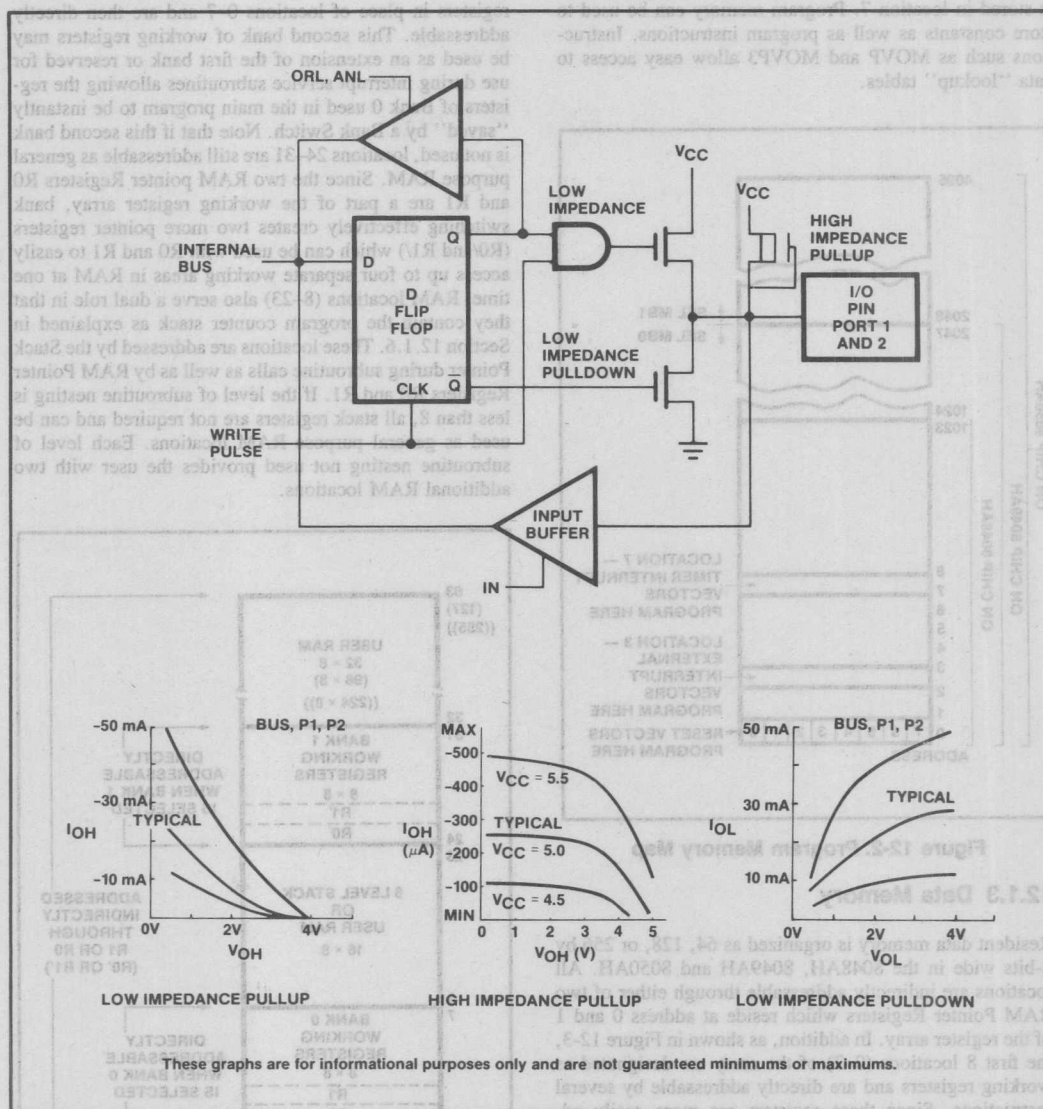


Figure 12-4. "Quasi-bidirectional" Port Structure

### 12.1.4 Input/Output

The 8048AH has 27 lines which can be used for input or output functions. These lines are grouped as 3 ports of 8 lines each which serve as either inputs, outputs or bidirectional ports and 3 "test" inputs which can alter program sequences when tested by conditional jump instructions.

#### PORTS 1 AND 2

Ports 1 and 2 are each 8 bits wide and have identical characteristics. Data written to these ports is statically latched and remains unchanged until rewritten. As input ports these lines are non-latching, i.e., inputs must be present until read by an input instruction. Inputs are fully TTL compatible and outputs will drive one standard TTL load.

The lines of ports 1 and 2 are called quasi-bidirectional because of a special output circuit structure which allows each line to serve as an input, and output, or both even though outputs are statically latched. Figure 12-4 shows the circuit configuration in detail. Each line is continuously pulled up to  $V_{CC}$  through a resistive device of relatively high impedance.

This pullup is sufficient to provide the source current for a TTL high level yet can be pulled low by a standard TTL gate thus allowing the same pin to be used for both input and output. To provide fast switching times in a "0" to "1" transition a relatively low impedance device is switched in momentarily ( $\approx 1/5$  of a machine cycle) whenever a "1" is written to the line. When a "0" is written to the line a low impedance device overcomes the light pullup and provides TTL current sinking capability. Since the pulldown transistor is a low impedance device a "1" must first be written to any line which is to be used as an input. Reset initializes all lines to the high impedance "1" state.

It is important to note that the ORL and the ANL are read/write operations. When executed, the  $\mu C$  "reads" the port, modifies the data according to the instruction, then "writes" the data back to the port. The "writing" (essentially an OUTL instruction) enables the low impedance pull-up momentarily again even if the data was unchanged from a "1." This specifically applies to configurations that have inputs and outputs mixed together on the same port. See also section 13.7.

#### BUS

Bus is also an 8-bit port which is a true bidirectional port with associated input and output strobes. If the bidirectional feature is not needed, Bus can serve as either a

statically latched output port or non-latching input port. Input and output lines on this port cannot be mixed however.

As a static port, data is written and latched using the OUTL instruction and inputted using the INS instruction. The INS and OUTL instructions generate pulses on the corresponding RD and WR output strobe lines; however, in the static port mode they are generally not used. As a bidirectional port the MOVX instructions are used to read and write the port. A write to the port generates a pulse on the WR output line and output data is valid at the trailing edge of WR. A read of the port generates a pulse on the RD output line and input data must be valid at the trailing edge of RD. When not being written or read, the BUS lines are in a high impedance state. See also sections 13.6 and 13.7.

### 12.1.5 Test and INT Inputs

Three pins serve as inputs and are testable with the conditional jump instruction. These are T0, T1, and INT. These pins allow inputs to cause program branches without the necessity to load an input port into the accumulator. The T0, T1, and INT pins have other possible functions as well. See the pin description in Section 12.2.

### 12.1.6 Program Counter and Stack

The Program Counter is an independent counter while the Program Counter Stack is implemented using pairs of registers in the Data Memory Array. Only 10, 11, or 12 bits of the Program Counter are used to address the 1024, 2048, or 4096 words of on-board program memory of the 8048AH, 8049AH, or 8050AH, while the most significant bits can be used for external Program Memory fetches. See Figure 12.5. The Program Counter is initialized to zero by activating the Reset line.

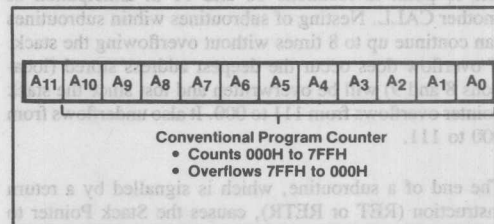


Figure 12-5. Program Counter

An interrupt or CALL to a subroutine causes the contents of the program counter to be stored in one of the 8 register pairs of the Program Counter Stack as shown in Figure 12-6. The pair to be used is determined by a 3-bit Stack Pointer which is part of the Program Status Word (PSW).



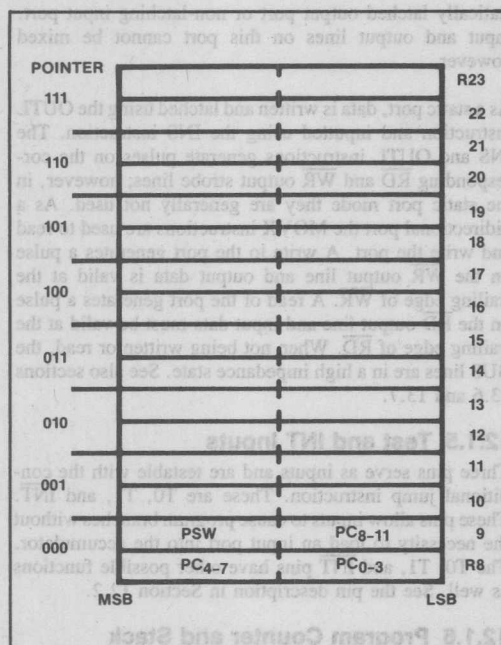


Figure 12-6. Program Counter Stack

Data RAM locations 8-23 are available as stack registers and are used to store the Program Counter and 4 bits of PSW as shown in Figure 12-6. The Stack Pointer when initialized to 000 points to RAM locations 8 and 9. The first subroutine jump or interrupt results in the program counter contents being transferred to locations 8 and 9 of the RAM array. The stack pointer is then incremented by one to point to locations 10 and 11 in anticipation of another CALL. Nesting of subroutines within subroutines can continue up to 8 times without overflowing the stack. If overflow does occur the deepest address stored (locations 8 and 9) will be overwritten and lost since the stack pointer overflows from 111 to 000. It also underflows from 000 to 111.

The end of a subroutine, which is signalled by a return instruction (RET or RETR), causes the Stack Pointer to be decremented and the contents of the resulting register pair to be transferred to the Program Counter.

### 12.1.7 Program Status Word

An 8-bit status word which can be loaded to and from the accumulator exists called the Program Status Word (PSW). Figure 12-7 shows the information available in

the word. The Program Status Word is actually a collection of flip-flops throughout the machine which can be read or written as a whole. The ability to write to PSW allows for easy restoration of machine status after a power down sequence.

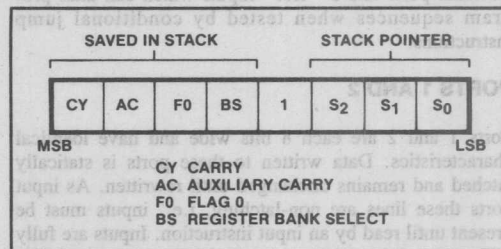


Figure 12-7. Program Status Word (PSW)

The upper four bits of PSW are stored in the Program Counter Stack with every call to subroutine or interrupt vector and are optionally restored upon return with the RETR instruction. The RET return instruction does not update PSW.

The PSW bit definitions are as follows:

- Bits 0-2: Stack Pointer bits ( $S_0, S_1, S_2$ )
- Bit 3: Not used ("1" level when read)
- Bit 4: Working Register Bank Switch Bit (BS)  
0 = Bank 0  
1 = Bank 1
- Bit 5: Flag 0 bit (F0) user controlled flag which can be complemented or cleared, and tested with the conditional jump instruction JF0.
- Bit 6: Auxiliary Carry (AC) carry bit generated by an ADD instruction and used by the decimal adjust instruction DA A.
- Bit 7: Carry (CY) carry flag which indicates that the previous operation has resulted in overflow of the accumulator.

### 12.1.8 Conditional Branch Logic

The conditional branch logic within the processor enables several conditions internal and external to the processor to be tested by the users program. By using the conditional jump instruction the conditions that are listed in Table 12-1 can effect a change in the sequence of the program execution.

Table 12-1

| Device Testable       | Jump Conditions<br>(Jump On) |               |
|-----------------------|------------------------------|---------------|
|                       | All zeros                    | not all zeros |
| Accumulator           | All zeros                    | not all zeros |
| Accumulator Bit       | —                            | 1             |
| Carry Flag            | 0                            | 1             |
| User Flags (F0, F1)   | —                            | 1             |
| Timer Overflow Flag   | —                            | 1             |
| Test Inputs (T0, T1)  | 0                            | 1             |
| Interrupt Input (INT) | 0                            | —             |

### 12.1.9 Interrupt

An interrupt sequence is initiated by applying a low "0" level input to the INT pin. Interrupt is level triggered and active low to allow "WIRE ORing" of several interrupt sources at the input pin. Figure 12-8 shows the interrupt logic of the 8048AH. The Interrupt line is sampled every instruction cycle and when detected causes a "call to subroutine" at location 3 in program memory as soon as all cycles of the current instruction are complete. On 2-cycle instructions the interrupt line is sampled on the 2nd cycle only. INT must be held low for at least 3 machine cycles to ensure proper interrupt operations. As in any CALL to subroutine, the Program Counter and Program Status word are saved in the stack. For a description of this operation see the previous section, Program Counter and Stack. Program Memory location 3 usually contains an unconditional jump to an interrupt service subroutine elsewhere in program memory. The end of an interrupt service subroutine is signalled by the execution of a Return and Restore Status instruction RETR. The interrupt system is single level in that once an interrupt is detected all further interrupt requests are ignored until execution of an RETR reenables the interrupt input logic. This occurs at the beginning of the second cycle of the RETR instruction. This sequence holds true also for an internal interrupt generated by timer overflow. If an internal timer/counter generated interrupt and an external interrupt are detected at the same time, the external source will be recognized. See the following Timer/Counter section for a description of timer interrupt. If needed, a second external interrupt can be created by enabling the timer/counter interrupt, loading FFH in the Counter (ones less than terminal count), and enabling the event counter mode. A "1" to "0" transition on the T1 input will then cause an interrupt vector to location 7.

### INTERRUPT TIMING

The interrupt input may be enabled or disabled under Program Control using the EN I and DIS I instructions. Interrupts are disabled by Reset and remain so until en-

abled by the users program. An interrupt request must be removed before the RETR instruction is executed upon return from the service routine otherwise the processor will re-enter the service routine immediately. Many peripheral devices prevent this situation by resetting their interrupt request line whenever the processor accesses (Reads or Writes) the peripherals data buffer register. If the interrupting device does not require access by the processor, one output line of the 8048AH may be designated as an "interrupt acknowledge" which is activated by the service subroutine to reset the interrupt request. The INT pin may also be tested using the conditional jump instruction JNL. This instruction may be used to detect the presence of a pending interrupt before interrupts are enabled. If interrupt is left disabled, INT may be used as another test input like T0 and T1.

### 12.1.10 Time/Counter

The 8048AH contains a counter to aid the user in counting external events and generating accurate time delays without placing a burden on the processor for these functions. In both modes the counter operation is the same, the only difference being the source of the input to the counter. The timer/event counter is shown in Figure 12-9.

### COUNTER

The 8-bit binary counter is presettable and readable with two MOV instructions which transfer the contents of the accumulator to the counter and vice versa. The counter content may be affected by Reset and should be initialized by software. The counter is stopped by a Reset or STOP TCNT instruction and remains stopped until started as a timer by a START T instruction or as an event counter by a START CNT instruction. Once started the counter will increment to this maximum count (FF) and overflow to zero continuing its count until stopped by a STOP TCNT instruction or Reset.

The increment from maximum count to zero (overflow) results in the setting of an overflow flag flip-flop and in the generation of an interrupt request. The state of the overflow flag is testable with the conditional jump instruction JTF. The flag is reset by executing a JTF or by Reset. The interrupt request is stored in a latch and then ORed with the external interrupt input INT. The timer interrupt may be enabled or disabled independently of external interrupt by the EN TCNT1 and DIS TCNT1 instructions. If enabled, the counter overflow will cause a subroutine call to location 7 where the timer or counter service routine may be stored.

If timer and external interrupts occur simultaneously, the external source will be recognized and the Call will be to

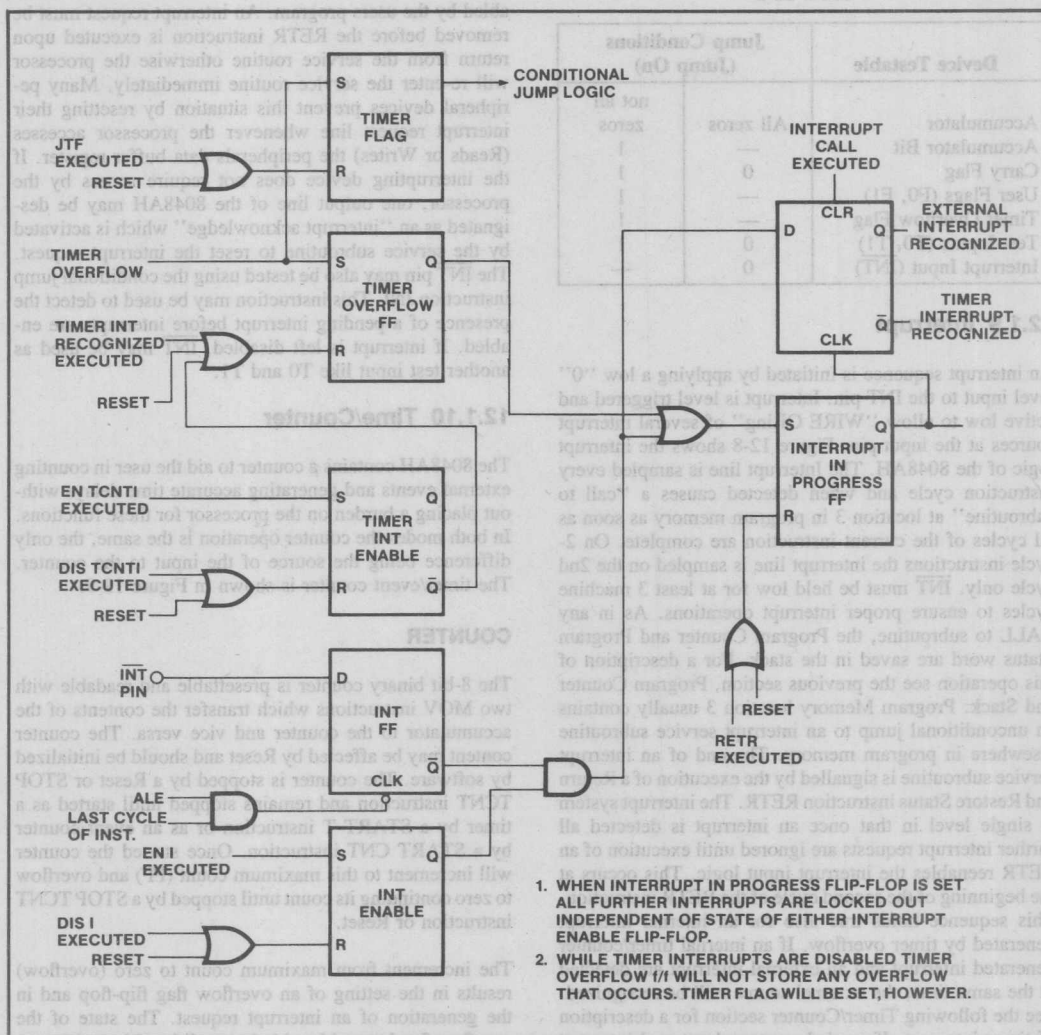


Figure 12-8. Interrupt Logic

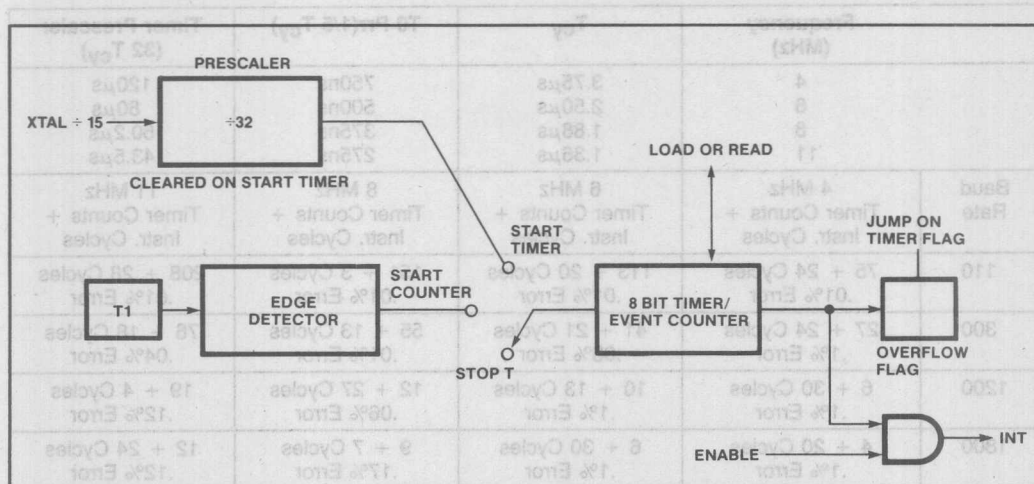


Figure 12-9. Timer/Event Counter

location 3. Since the timer interrupt is latched it will remain pending until the external device is serviced and immediately be recognized upon return from the service routine. The pending timer interrupt is reset by the Call to location 7 or may be removed by executing a DIS TCNT1 instruction.

#### AS AN EVENT COUNTER

Execution of a START CNT instruction connects the T1 input pin to the counter input and enables the counter. The T1 input is sampled at the beginning of state 3 or in later MCS-48 devices in state time 4. Subsequent high to low transitions on T1 will cause the counter to increment. T1 must be held low for at least 1 machine cycle to insure it won't be missed. The maximum rate at which the counter may be incremented is once per three instruction cycles (every 5.7  $\mu$ sec when using an 8 MHz crystal)—there is no minimum frequency. T1 input must remain high for at least 1/5 machine cycle after each transition.

#### AS A TIMER

Execution of a START T instruction connects an internal clock to the counter input and enables the counter. The internal clock is derived bypassing the basic machine cycle clock through a  $\div 32$  prescaler. The prescaler is reset during the START T instruction. The resulting clock increments the counter every 32 machine cycles. Various delays from 1 to 256 counts can be obtained by presetting the counter and detecting overflow. Times longer than 256 counts may be achieved by accumulating multiple overflows in a register under software control. For time res-

olution less than 1 count an external clock can be applied to the T1 input and the counter operated in the event counter mode. ALE divided by 3 or more can serve as this external clock. Very small delays or "fine tuning" of larger delays can be easily accomplished by software delay loops.

Often a serial link is desirable in an MCS-48 family member. Table 12-2 lists the timer counts and cycles needed for a specific baud rate given a crystal frequency.

#### 12.1.11 Clock and Timing Circuits

Timing generation for the 8048AH is completely self-contained with the exception of a frequency reference which can be XTAL, ceramic resonator, or external clock source. The Clock and Timing circuitry can be divided into the following functional blocks.

##### OSCILLATOR

The on-board oscillator is a high gain parallel resonant circuit with a frequency range of 1 to 11 MHz. The X1 external pin is the input to the amplifier stage while X2 is the output. A crystal or ceramic resonator connected between X1 and X2 provides the feedback and phase shift required for oscillation. If an accurate frequency reference is not required, ceramic resonator may be used in place of the crystal.

For accurate clocking, a crystal should be used. An externally generated clock may also be applied to X1-X2 as the frequency source. See the data sheet for more information.



Table 12-2. Baud Rate Generation

|              | Frequency<br>(MHz)                       | T <sub>cy</sub>                          | T0 Prr(1/5 T <sub>cy</sub> )             | Timer Prescaler<br>(32 T <sub>cy</sub> )  |
|--------------|--|--|--|---|
|              | 4  | 3.75μs                                   | 750ns                                    | 120μs                                     |
|              | 6  | 2.50μs                                   | 500ns                                    | 80μs                                      |
|              | 8  | 1.88μs                                   | 375ns                                    | 60.2μs                                    |
|              | 11                                       | 1.36μs                                   | 275ns                                    | 43.5μs                                    |
| Baud<br>Rate | 4 MHz<br>Timer Counts +<br>Instr. Cycles | 6 MHz<br>Timer Counts +<br>Instr. Cycles | 8 MHz<br>Timer Counts +<br>Instr. Cycles | 11 MHz<br>Timer Counts +<br>Instr. Cycles |
| 110          | 75 + 24 Cycles<br>.01% Error             | 113 + 20 Cycles<br>.01% Error            | 151 + 3 Cycles<br>.01% Error             | 208 + 28 Cycles<br>.01% Error             |
| 300          | 27 + 24 Cycles<br>.1% Error              | 41 + 21 Cycles<br>.03% Error             | 55 + 13 Cycles<br>.01% Error             | 76 + 18 Cycles<br>.04% Error              |
| 1200         | 6 + 30 Cycles<br>.1% Error               | 10 + 13 Cycles<br>.1% Error              | 12 + 27 Cycles<br>.06% Error             | 19 + 4 Cycles<br>.12% Error               |
| 1800         | 4 + 20 Cycles<br>.1% Error               | 6 + 30 Cycles<br>.1% Error               | 9 + 7 Cycles<br>.17% Error               | 12 + 24 Cycles<br>.12% Error              |
| 2400         | 3 + 15 Cycles<br>.1% Error               | 5 + 6 Cycles<br>.4% Error                | 6 + 24 Cycles<br>.29% Error              | 9 + 18 Cycles<br>.12% Error               |
| 4800         | 1 + 23 Cycles<br>1.0% Error              | 2 + 19 Cycles<br>.4% Error               | 3 + 14 Cycles<br>.74% Error              | 4 + 25 Cycles<br>.12% Error               |

## STATE COUNTER

The output of the oscillator is divided by 3 in the State Counter to create a clock which defines the state times of the machine (CLK). CLK can be made available on the external pin T0 by executing an ENTO CLK instruction. The output of CLK on T0 is disabled by Reset of the processor.

## CYCLE COUNTER

CLK is then divided by 5 in the Cycle Counter to provide a clock which defines a machine cycle consisting of 5 machine states as shown in Figure 12-10. Figure 12-11 shows the different internal operations as divided into the machine states. This clock is called Address Latch Enable (ALE) because of its function in MCS-48 systems with external memory. It is provided continuously on the ALE output pin.

### 12.1.12 Reset

The reset input provides a means for initialization for the processor. This Schmitt-trigger input has an internal pull-up device which in combination with an external 1 μf capacitor provides an internal reset pulse of sufficient length to guarantee all circuitry is reset, as shown in Figure 12-12. If the reset pulse is generated externally the RESET pin must be held low for at least 10 milliseconds after the

power supply is within tolerance. Only 5 machine cycles (6.8 μs @ 11 MHz) are required if power is already on and the oscillator has stabilized. ALE and PSEN (if EA = 1) are active while in Reset.

Reset performs the following functions:

- 1) Sets program counter to zero.
- 2) Sets stack pointer to zero.
- 3) Selects register bank 0.
- 4) Selects memory bank 0.
- 5) Sets BUS to high impedance state (except when EA = 5V).
- 6) Sets Ports 1 and 2 to input mode.
- 7) Disables interrupts (timer and external).
- 8) Stops timer.
- 9) Clears timer flag.
- 10) Clears F0 and F1.
- 11) Disables clock output from T0.

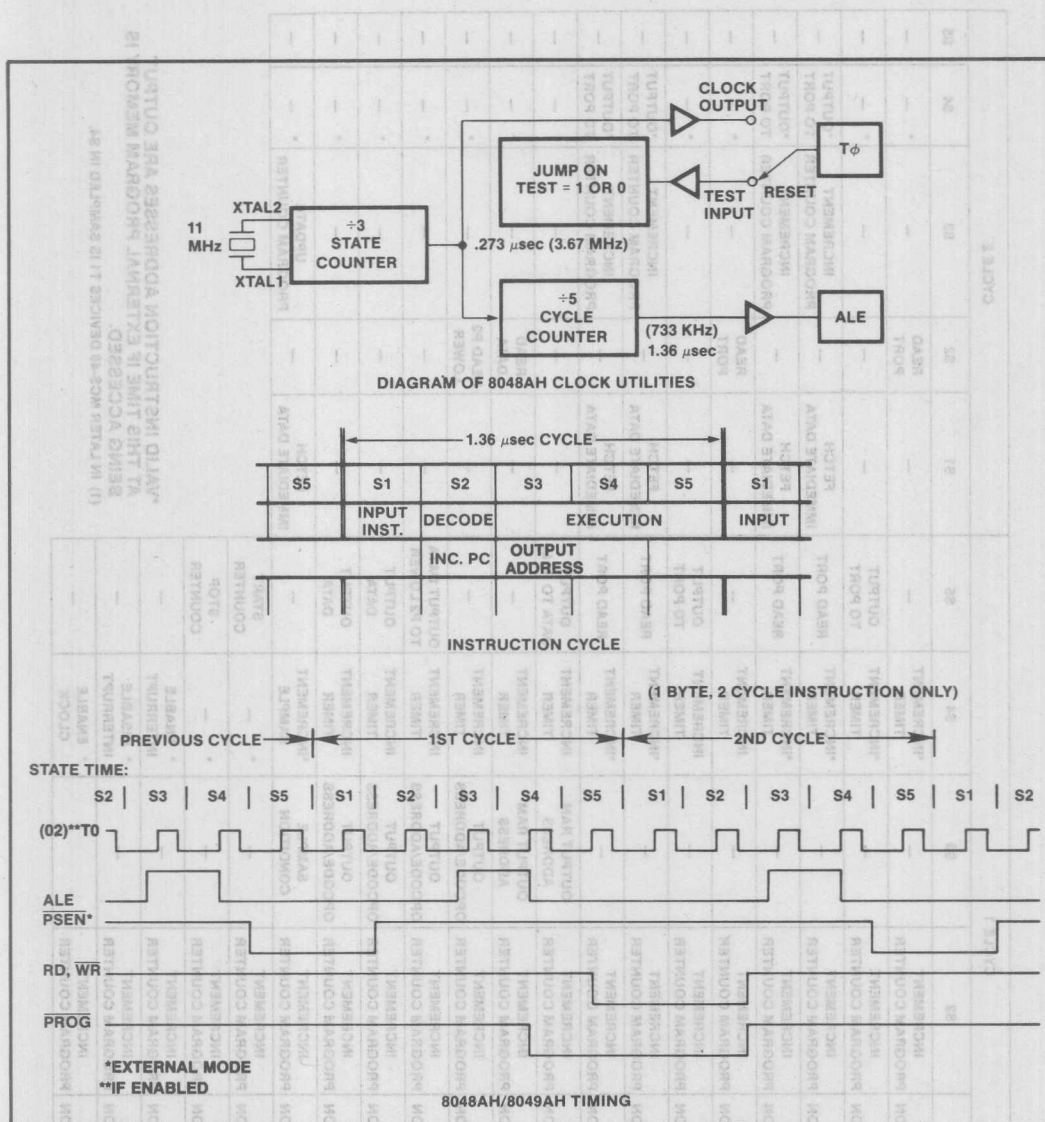


Figure 12-10. MCS®-48 Timing Generation and Cycle Timing

### 12.1.13 Single-Step

This feature, as pictured in Figure 12-13, provides the user with a debug capability in that the processor can be stepped through the program one instruction at a time. While stopped, the address of the next instruction to be fetched is available concurrently on BUS and the lower

half of Port 2. The user can therefore follow the program through each of the instruction steps. A timing diagram, showing the interaction between output ALE and input SS, is shown. The BUS buffer contents are lost during single step; however, a latch may be added to reestablish the lost I/O capability if needed. Data is valid at the leading edge of ALE.

Figure 12-11. 8048AH/8049AH Instruction Timing Diagram

| INSTRUCTION            | CYCLE 1           |                           |                        |                     |                         | CYCLE 2  |               |                           |                 |    |
|------------------------|-------------------|---------------------------|------------------------|---------------------|-------------------------|--|---------------|---------------------------|-----------------|----|
|                        | S1                | S2                        | S3                     | S4                  | S5                      | S1   | S2            | S3                        | S4              | S5 |
| IN A,P                 | FETCH INSTRUCTION | INCREMENT PROGRAM COUNTER | —                      | *INCREMENT TIMER    | —                       | —  | READ PORT     | —                         | * —             | —  |
| OUTL P,A               | FETCH INSTRUCTION | INCREMENT PROGRAM COUNTER | —                      | *INCREMENT TIMER    | OUTPUT TO PORT          | —  | —             | —                         | * —             | —  |
| ANL P, = DATA          | FETCH INSTRUCTION | INCREMENT PROGRAM COUNTER | —                      | *INCREMENT TIMER    | READ PORT               | FETCH IMMEDIATE DATA   | —             | INCREMENT PROGRAM COUNTER | *OUTPUT TO PORT | —  |
| ORL P, = DATA          | FETCH INSTRUCTION | INCREMENT PROGRAM COUNTER | —                      | *INCREMENT TIMER    | READ PORT               | FETCH IMMEDIATE DATA   | —             | INCREMENT PROGRAM COUNTER | *OUTPUT TO PORT | —  |
| INS A, BUS             | FETCH INSTRUCTION | INCREMENT PROGRAM COUNTER | —                      | INCREMENT TIMER     | —                       | —  | READ PORT     | —                         | * —             | —  |
| OUTL BUS, A            | FETCH INSTRUCTION | INCREMENT PROGRAM COUNTER | —                      | INCREMENT TIMER     | OUTPUT TO PORT          | —  | —             | —                         | * —             | —  |
| ANL BUS, = DATA        | FETCH INSTRUCTION | INCREMENT PROGRAM COUNTER | —                      | *INCREMENT TIMER    | READ PORT               | FETCH IMMEDIATE DATA   | —             | INCREMENT PROGRAM COUNTER | *OUTPUT TO PORT | —  |
| ORL BUS, = DATA        | FETCH INSTRUCTION | INCREMENT PROGRAM COUNTER | —                      | *INCREMENT TIMER    | READ PORT               | FETCH IMMEDIATE DATA   | —             | INCREMENT PROGRAM COUNTER | *OUTPUT TO PORT | —  |
| MOVX @ R,A             | FETCH INSTRUCTION | INCREMENT PROGRAM COUNTER | OUTPUT RAM ADDRESS     | INCREMENT TIMER     | OUTPUT DATA TO RAM      | —  | —             | —                         | * —             | —  |
| MOVX A,@R              | FETCH INSTRUCTION | INCREMENT PROGRAM COUNTER | OUTPUT RAM ADDRESS     | INCREMENT TIMER     | —                       | —  | READ DATA     | —                         | * —             | —  |
| MOVD A,P <sub>i</sub>  | FETCH INSTRUCTION | INCREMENT PROGRAM COUNTER | OUTPUT OP CODE/ADDRESS | INCREMENT TIMER     | —                       | —  | READ P2 LOWER | —                         | * —             | —  |
| MOVD P <sub>i</sub> ,A | FETCH INSTRUCTION | INCREMENT PROGRAM COUNTER | OUTPUT OP CODE/ADDRESS | INCREMENT TIMER     | OUTPUT DATA TO P2 LOWER | —  | —             | —                         | * —             | —  |
| ANLD P,A               | FETCH INSTRUCTION | INCREMENT PROGRAM COUNTER | OUTPUT OP CODE/ADDRESS | INCREMENT TIMER     | OUTPUT DATA             | —  | —             | —                         | * —             | —  |
| ORLD P,A               | FETCH INSTRUCTION | INCREMENT PROGRAM COUNTER | OUTPUT OP CODE/ADDRESS | INCREMENT TIMER     | OUTPUT DATA             | —  | —             | —                         | * —             | —  |
| J(CONDITIONAL)         | FETCH INSTRUCTION | INCREMENT PROGRAM COUNTER | SAMPLE CONDITION       | *INCREMENT SAMPLE   | —                       | FETCH IMMEDIATE DATA   | —             | UPDATE PROGRAM COUNTER    | * —             | —  |
| STRT T                 | FETCH INSTRUCTION | INCREMENT PROGRAM COUNTER | —                      | * —                 | START COUNTER           | *VALID INSTRUCTION ADDRESSES ARE OUTPUT AT THIS TIME IF EXTERNAL PROGRAM MEMORY IS BEING ACCESSED.<br>(1) IN LATER MCS-48 DEVICES T1 IS SAMPLED IN S4. |               |                           |                 |    |
| STOP TCNT              | FETCH INSTRUCTION | INCREMENT PROGRAM COUNTER | —                      | * —                 | STOP COUNTER            |  |               |                           |                 |    |
| ENI                    | FETCH INSTRUCTION | INCREMENT PROGRAM COUNTER | —                      | * ENABLE INTERRUPT  | —                       |  |               |                           |                 |    |
| DIS I                  | FETCH INSTRUCTION | INCREMENT PROGRAM COUNTER | —                      | * DISABLE INTERRUPT | —                       |  |               |                           |                 |    |
| ENTO CLK               | FETCH INSTRUCTION | INCREMENT PROGRAM COUNTER | —                      | * ENABLE CLOCK      | —                       |  |               |                           |                 |    |

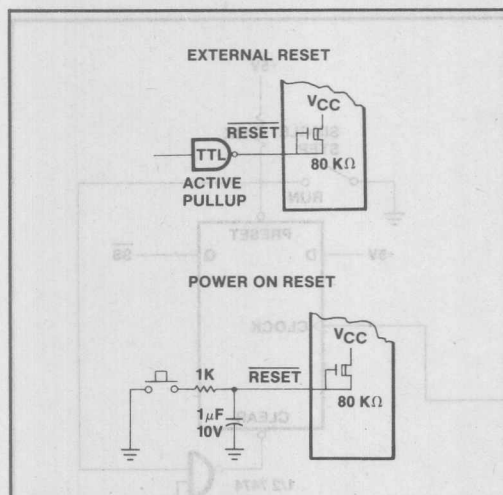


Figure 12-12

## TIMING

The 8048AH operates in a single-step mode as follows:

- 1) The processor is requested to stop by applying a low level on  $\overline{SS}$ .
- 2) The processor responds by stopping during the address fetch portion of the next instruction. If a double cycle instruction is in progress when the single step command is received, both cycles will be completed before stopping.
- 3) The processor acknowledges it has entered the stopped state by raising ALE high. In this state (which can be maintained indefinitely) the address of the next instruction to be fetched is present on BUS and the lower half of port 2.
- 4)  $\overline{SS}$  is then raised high to bring the processor out of the stopped mode allowing it to fetch the next instruction. The exit from stop is indicated by the processor bringing ALE low.
- 5) To stop the processor at the next instruction  $\overline{SS}$  must be brought low again soon after ALE goes low. If  $\overline{SS}$  is left high the processor remains in a "Run" mode.

A diagram for implementing the single-step function of the 8748H is shown in Figure 12-13. D-type flip-flop with preset and clear is used to generate  $\overline{SS}$ . In the run mode  $\overline{SS}$  is held high by keeping the flip-flop preset (preset has precedence over the clear input). To enter single step, preset is removed allowing ALE to bring  $\overline{SS}$  low via the

clear input. ALE should be buffered since the clear input of an SN7474 is the equivalent of 3 TTL loads. The processor is now in the stopped state. The next instruction is initiated by clocking a "1" into the flip-flop. This "1" will not appear on  $\overline{SS}$  unless ALE is high removing clear from the flip-flop. In response to  $\overline{SS}$  going high the processor begins an instruction fetch which brings ALE low resetting  $\overline{SS}$  through the clear input and causing the processor to again enter the stopped state.

#### 12.1.14 Power Down Mode (8048AH, 8049AH, 8050AH, 8039AHL, 8035AHL, 8040AHL)

Extra circuitry has been added to the 8048AH/8049AH/8050AH ROM version to allow power to be removed from all but the data RAM array for low power standby operation. In the power down mode the contents of data RAM can be maintained while drawing typically 10% to 15% of normal operating power requirements.

$V_{CC}$  serves as the 5V supply pin for the bulk of circuitry while the  $V_{DD}$  pin supplies only the RAM array. In normal operation both pins are a 5V while in standby,  $V_{CC}$  is at ground and  $V_{DD}$  is maintained at its standby value. Applying Reset to the processor through the RESET pin inhibits any access to the RAM by the processor and guarantees that RAM cannot be inadvertently altered as power is removed from  $V_{CC}$ .

A typical power down sequence (Figure 12-14) occurs as follows:

- 1) Imminent power supply failure is detected by user defined circuitry. Signal must be early enough to allow 8048AH to save all necessary data before  $V_{CC}$  falls below normal operating limits.
- 2) Power fail signal is used to interrupt processor and vector it to a power fail service routine.
- 3) Power fail routine saves all important data and machine status in the internal data RAM array. Routine may also initiate transfer of backup supply to the  $V_{DD}$  pin and indicate to external circuitry that power fail routine is complete.
- 4) Reset is applied to guarantee data will not be altered as the power supply falls out of limits. Reset must be held low until  $V_{CC}$  is at ground level.

Recovery from the Power Down mode can occur as any other power-on sequence with an external capacitor on the Reset input providing the necessary delay. See the previous section on Reset.



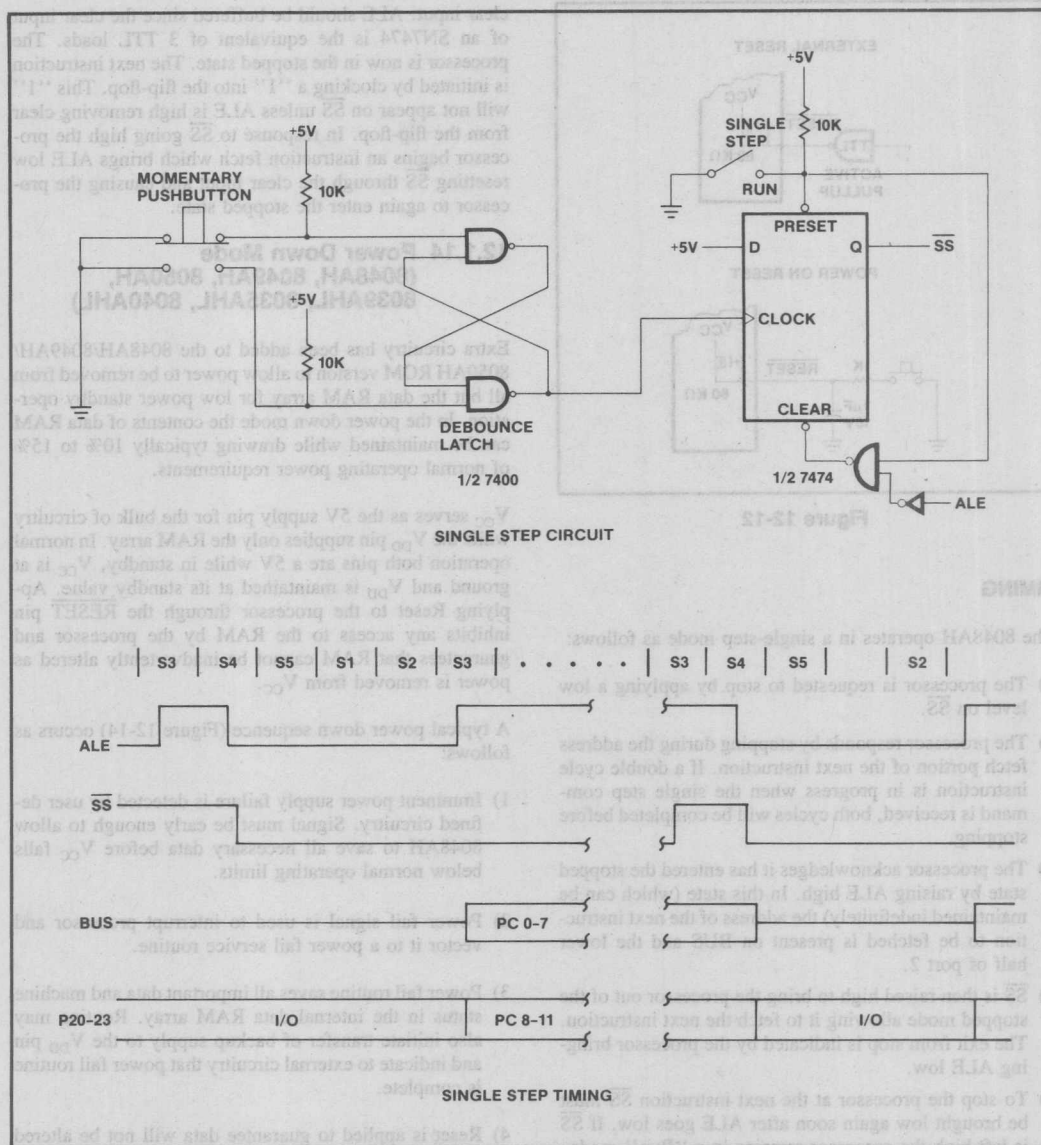


Figure 12-13. Single Step Operation

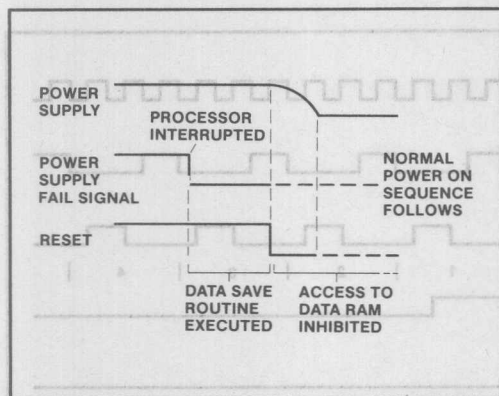


Figure 12-14. Power Down Sequence

### 12.1.15 External Access Mode

Normally the first 1K (8048AH), 2K (8049AH), or 4K (8050AH) words of program memory are automatically fetched from internal ROM or EPROM. The EA input pin however allows the user to effectively disable internal program memory by forcing all program memory fetches to reference external memory. The following chapter explains how access to external program memory is accomplished.

The External Access mode is very useful in system test and debug because it allows the user to disable his internal applications program and substitute an external program of his choice — a diagnostic routine for instance. In addition, section 12.4 explains how internal program memory can be read externally, independent of the processor. A "1" level on EA initiates the external access mode. For proper operation, Reset should be applied while the EA input is changed.

### 12.1.16 Sync Mode

The 8048AH, 8049AH, 8050AH has incorporated a new SYNC mode. The Sync mode is provided to ease the design of multiple controller circuits by allowing the designer to force the device into known phase and state time. The SYNC mode may also be utilized by automatic test equipment (ATE) for quick, easy, and efficient synchronizing between the tester and the DUT (device under test).

SYNC mode is enabled when SS' pin is raised to high voltage level of +12 volts. To begin synchronization, T0 is raised to 5 volts at least four clocks cycles after SS'. T0 must be high for at least four X1 clock cycles to fully

reset the prescaler and time state generators. T0 may then be brought down with the rising edge of X1. Two clock cycles later, with the rising edge of X1, the device enters into Time State 1, Phase 1, SS' is then brought down to 5 volts 4 clocks later after T0. RESET' is allowed to go high 5 tCY (75 clocks) later for normal execution of code. See Figure 12-15.

### 12.1.17 Idle Mode

Along with the standard power down, the 80C438, 80C49, 80C50 has added an IDLE mode instruction (01H) to give even further flexibility and power management. In the IDLE mode, the CPU is frozen while the oscillator, RAM, timer, and the interrupt circuitry remains fully active.

When the IDL instruction (01H) is decoded, the clock to the CPU is stopped. CPU status is preserved in its entirety: the Stack Pointer, Program Counter, Program Status Word, Accumulator, RAM, and all the registers maintain their data throughout idle.

Externally, the following occurs during idle:

- 1) The ports remain in the logical state they were in when idle was executed.
- 2) The bus remains in the logical state it was in when idle was executed if the bus was latched.  
If the bus was in a high Z condition or if external program memory is used the bus will remain in the float state.
- 3) ALE remains in the inactive state (low).
- 4) RD', WR', PROG', and PSEN' remains in the inactive state (high).
- 5) T0 outputs clock if enabled.

There are three ways of exiting idle. Activating any enabled interrupt (external or timer) will cause the CPU to vector to the appropriate interrupt routine. Following a RETR instruction, program execution will resume at the instruction following the address that contained the IDL instruction.

The F0 and F1 flags may be used to give an indication if the interrupt occurred during normal program execution or during idle. This is done by setting or clearing the flags before going into idle. The interrupt service routine can examine the flags and act accordingly when idle is terminated by an interrupt.

Resetting the device can also terminate idle. Since the oscillator is already running, five machine cycles are all that is required to insure proper machine operation.

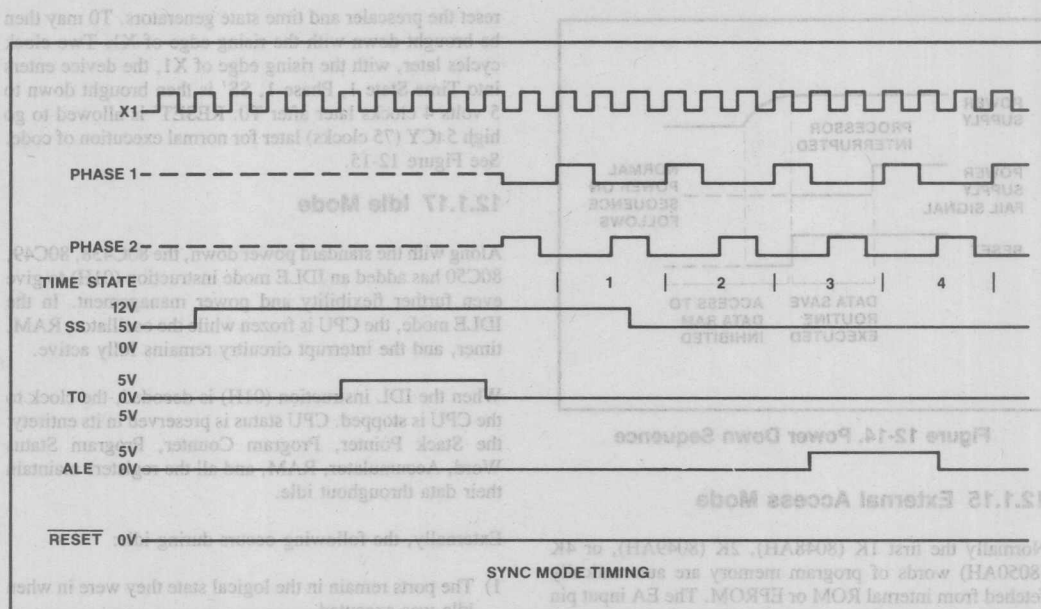


Figure 12-15. Sync Mode Timing

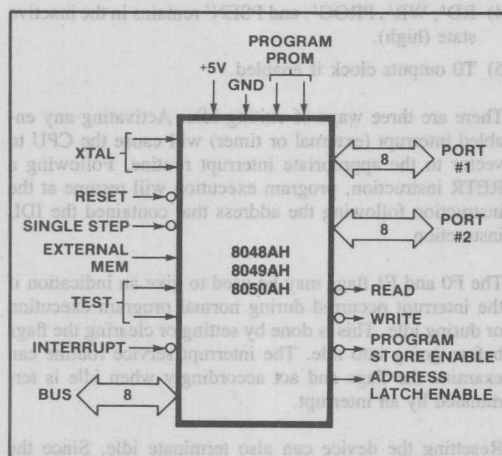


Figure 12-16. 8048AH and 8049AH Logic Symbol

## 12.2 PIN DESCRIPTION

The MCS-48 processors are packaged in 40 pin Dual In-Line Packages (DIP's). Table 12-3 is a summary of the functions of each pin. Figure 12-16 is the logic symbol for the 8048AH product family. Where it exists, the second paragraph describes each pin's function in an expanded MCS-48 system. Unless otherwise specified, each input is TTL compatible and each output will drive one standard TTL load.

SYNC mode is enabled when SS<sup>1</sup> pin is raised to high voltage level of +12 volts. To begin synchronization, TO is raised to 5 volts at least four clock cycles after SS<sup>1</sup>. TO must be high for at least four X1 clock cycles to fully

# SINGLE COMPONENT MCS®-48 SYSTEM

Table 12-3. Pin Description

| Designation         | Pin Number*    | Function   |
|---------------------|----------------|--|
| V <sub>SS</sub>     | 20             | Circuit GND potential  |
| V <sub>DD</sub>     | 26             | Programming power supply; 21V during program for the 8748H/8749H; +5V during operation for both ROM and EPROM. Low power standby pin in 8048AH and 8049AH/8050AH ROM versions.   |
| V <sub>CC</sub>     | 40             | Main power supply; +5V during operation and during 8748H and 8749H programming.  |
| PROG                | 25             | Program pulse; +18V input pin during 8748H/8749H programming. Output strobe for 8243 I/O expander.   |
| P10-P17<br>(Port 1) | 27-34          | 8-bit quasi-bidirectional port. (Internal Pullup $\approx 50K\Omega$ )   |
| P20-P27<br>(Port 2) | 21-24<br>35-38 | 8-bit quasi-bidirectional port. (Internal Pullup $\approx 50K\Omega$ )<br><br>P20-P23 contain the four high order program counter bits during an external program memory fetch and serve as a 4-bit I/O expander bus for 8243.   |
| D0-D7<br>(BUS)      | 12-19          | True bidirectional port which can be written or read synchronously using the $\overline{RD}$ , $\overline{WR}$ strobes. The port can also be statically latched.<br><br>Contains the 8 low order program counter bits during an external program memory fetch, and receives the addressed instruction under the control of $\overline{PSEN}$ . Also contains the address and data during an external RAM data store instruction, under control of $\overline{ALE}$ , $\overline{RD}$ , and $\overline{WR}$ . |
| T0                  | 1              | Input pin testable using the conditional transfer instructions JT0 and JNT0. T0 can be designated as a clock output using ENTO CLK instruction. T0 is also used during programming and sync mode.  |
| T1                  | 39             | Input pin testable using the JT1, and JNT1 instructions. Can be designated the event counter input using the STRT CNT instruction. (See Section 2.1.10)  |
| INT                 | 6              | Interrupt input. Initiates an interrupt if interrupt is enabled. Interrupt is disabled after a reset. (Active low)<br><br>Interrupt must remain low for at least 3 machine cycles to ensure proper operation.  |
| $\overline{RD}$     | 8              | Output strobe activated during a BUS read. Can be used to enable data onto the BUS from an external device. (Active low)<br><br>Used as a Read Strobe to External Data Memory.   |
| $\overline{RESET}$  | 4              | Input which is used to initialize the processor. Also used during EPROM programming and verification. (Active low) (Internal pullup $\approx 80K\Omega$ )  |
| $\overline{WR}$     | 10             | Output strobe during a BUS write. (Active low) Used as write strobe to external data memory.   |
| $\overline{ALE}$    | 11             | Address Latch Enable. This signal occurs once during each cycle and is useful as a clock output.<br><br>The negative edge of $\overline{ALE}$ strobes address into external data and program memory.   |



Table 12-3. Pin Description (Continued)

| Designation | Pin Number* | Function   |
|-------------|-------------|--|
| PSEN        | 9           | Program Store Enable. This output occurs only during a fetch to external program memory. (Active low)  |
| SS          | 5           | Single step input can be used in conjunction with ALE to "single step" the processor through each instruction. (Active low) (Internal pullup $\approx 300K\Omega$ ) +12V for sync modes (See 2.1.16)   |
| EA          | 7           | External Access input which forces all program memory fetches to reference external memory. Useful for emulation and debug, and essential for testing and program verification. (Active high) +12V for 8048AH/8049AH/8050AH program verification and +18V for 8748H/8749H program verification (Internal pullup $\approx 10M\Omega$ on 8048AH/8049AH/8035AHL/8039AHL/8050AH/8040AHL) |
| XTAL1       | 2           | One side of crystal input for internal oscillator. Also input for external source.   |
| XTAL2       | 3           | Other side of crystal/external source input.   |

\*Unless otherwise stated, inputs do not have internal pullup resistors. 8048AH, 8748H, 8049AH, 8050AH, 8040AHL

## 12.3 PROGRAMMING, VERIFYING AND ERASING EPROM

The internal Program Memory of the 8748H and the 8749H may be erased and reprogrammed by the user as explained in the following sections. See also the 8748H and 8749H data sheets.

### 12.3.1 Programming/Verification

In brief, the programming process consists of: activating the program mode, applying an address, latching the address, applying data, and applying a programming pulse. This programming algorithm applies to both the 8748H and 8749H. Each word is programmed completely before moving on to the next and is followed by a verification step. The following is a list of the pins used for programming and a description of their functions:

| Pin             | Function   |
|-----------------|--|
| XTAL 1          | Clock Input (3 to 4 MHz)                         |
| Reset           | Initialization and Address Latching              |
| Test 0          | Selection of Program (0V) or Verify (5V) Mode    |
| EA              | Activation of Program/Verify Modes               |
| BUS             | Address and Data Input Data Output During Verify |
| P20-1           | Address Input for 8748H                          |
| P20-2           | Address Input for 8749H                          |
| V <sub>DD</sub> | Programming Power Supply                         |
| PROG            | Program Pulse Input                              |
| P10-P11         | Tied to ground (8749H only)                      |

## 8748H AND 8749H ERASURE CHARACTERISTICS

The erasure characteristics of the 8748H and 8749H are such that erasure begins to occur when exposed to light with wavelengths shorter than approximately 4000 Angstroms (Å). It should be noted that sunlight and certain types of fluorescent lamps have wavelengths in the 3000–4000Å range. Data show that constant exposure to room level fluorescent lighting could erase the typical 8748H and 8749H in approximately 3 years while it would take approximately 1 week to cause erasure when exposed to direct sunlight. If the 8748H or 8749H is to be exposed to these types of lighting conditions for extended periods of time, opaque labels should be placed over the 8748H window to prevent unintentional erasure.

When erased, bits of the 8748H and 8749H Program Memory are in the logic "0" state.

The recommended erasure procedure for the 8748H and 8749H is exposure to shortwave ultraviolet light which has a wavelength of 2537 Angstroms (Å). The integrated dose (i.e., UV intensity X exposure time) for erasure should be a minimum of 15W-sec/cm<sup>2</sup>. The erasure time with this dosage is approximately 15 to 20 minutes using an ultraviolet lamp with a 12000μW/cm<sup>2</sup> power rating. The 8748H and 8749H should be placed within one inch from the lamp tubes during erasure. Some lamps have a filter in their tubes and this filter should be removed before erasure.

# SINGLE COMPONENT MCS-48 SYSTEM

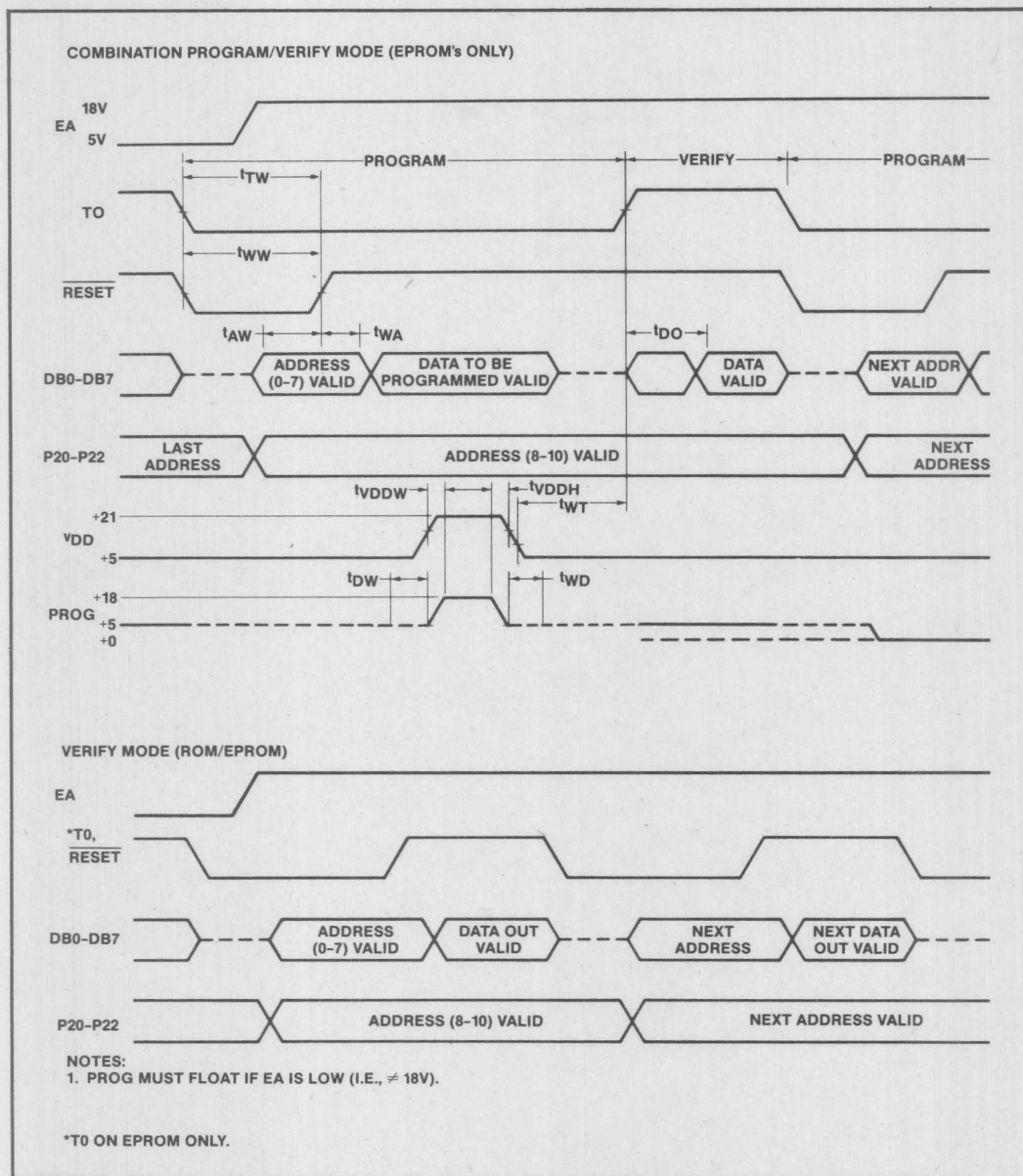


Figure 12-17. Program/Verify Sequence for 8749H/8748H









# CHAPTER 13

## EXPANDED MCS<sup>®</sup>-48 SYSTEM

### 13.0 INTRODUCTION

If the capabilities resident on the single-chip 8048AH/8748H/8035AHL/8049AH/8749H/8039AHL are not sufficient for your system requirements, special on-board circuitry allows the addition of a wide variety of external memory, I/O, or special peripherals you may require. The processors can be directly and simply expanded in the following areas:

- Program Memory to 4K words
- Data Memory to 320 words (384 words with 8049AH)
- I/O by unlimited amount
- Special Functions using 8080/8085AH peripherals

By using bank switching techniques, maximum capability is essentially unlimited. Bank switching is discussed later in the chapter. Expansion is accomplished in two ways:

- 1) Expander I/O — A special I/O Expander circuit, the 8243, provides for the addition of four 4-bit Input/Output ports with the sacrifice of only the lower half (4-bits) of port 2 for inter-device communication. Multiple 8243's may be added to this 4-bit bus by generating the required "chip select" lines.
- 2) Standard 8085 Bus — One port of the 8048AH/8049AH is like the 8-bit bidirectional data bus of the 8085 microcomputer system allowing interface to the numerous standard memories and peripherals of the MCS<sup>®</sup>-80/85 microcomputer family.

MCS-48 systems can be configured using either or both of these expansion features to optimize system capabilities to the application.

Both expander devices and standard memories and peripherals can be added in virtually any number and combination required.

### 13.1 EXPANSION OF PROGRAM MEMORY

Program Memory is expanded beyond the resident 1K or 2K words by using the 8085 BUS feature of the MCS<sup>®</sup>-48. All program memory fetches from the addresses less than 1024 on the 8048AH and less than 2048 on the 8049AH occur internally with no external signals being generated (except ALE which is always present). At address 1024 on the 8048AH, the processor automatically initiates external program memory fetches.

#### 13.1.1 Instruction Fetch Cycle (External)

As shown in Figure 13-1, for all instruction fetches from addresses of 1024 (2048) or greater, the following will occur:

- 1) The contents of the 12-bit program counter will be output on BUS and the lower half of port 2.
- 2) Address Latch Enable (ALE) will indicate the time at which address is valid. The trailing edge of ALE is used to latch the address externally.
- 3) Program Store Enable ( $\overline{\text{PSEN}}$ ) indicates that an external instruction fetch is in progress and serves to enable the external memory device.
- 4) BUS reverts to input (floating) mode and the processor accepts its 8-bit contents as an instruction word.

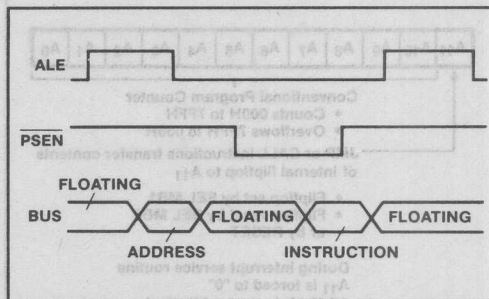


Figure 13-1. Instruction Fetch from External Program Memory

All instruction fetches, including internal addresses, can be forced to be external by activating the EA pin of the 8048AH/8049AH/8050AH. The 8035AHL/8039AHL/8040AHL processors without program memory always operate in the external program memory mode (EA = 5V).

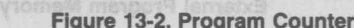
#### 13.1.2 Extended Program Memory Addressing (Beyond 2K)

For programs of 2K words or less, the 8048AH/8049AH addresses program memory in the conventional manner. Addresses beyond 2047 can be reached by executing a program memory bank switch instruction (SEL MB0, SEL MB1) followed by a branch instruction (JMP or CALL). The bank switch feature extends the range of branch instructions beyond their normal 2K range and at the same time prevents the user from inadvertently crossing the 2K boundary.

#### PROGRAM MEMORY BANK SWITCH

The switching of 2K program memory banks is accomplished by directly setting or resetting the most significant bit of the program counter (bit 11); see Figure 13-2. Bit 11 is not altered by normal incrementing of the program counter but is loaded with the contents of a special flip-flop each time a JMP or CALL instruction is executed. This special flip-flop is set by executing an SEL MB1

4) BUS reverts to input (floating) mode and the processor



## INTERRUPT ROUTINES

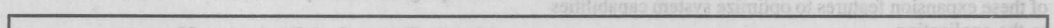
## INTERVIEW PROTOCOLS

## mation

802410

### 13.1.4 Expansion Examples

#### 15.1.4 Expansion Examples



**Figure 13-3. Expanding MCS®-48 Program Memory Using Standard Memory Products**

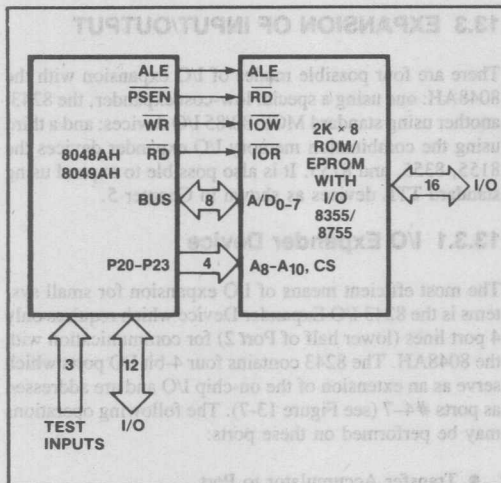


Figure 13-4. External Program Memory Interface

Figure 13-4 shows how the 8755/8355 EPROM/ROM with I/O interfaces directly to the 8048AH without the need for an address latch. The 8755/8355 contains an internal 8-bit address latch eliminating the need for an 8212 latch. In addition to a 2K x 8 program memory, the 8755/8355 also contains 16 I/O lines addressable as two 8-bit ports. These ports are addressed as external RAM; therefore the RD and WR outputs of the 8048AH are required. See the following section on data memory expansion for more detail. The subsequent section on I/O expansion explains the operation of the 16 I/O lines.

## 13.2 EXPANSION OF DATA MEMORY

Data Memory is expanded beyond the resident 64 words by using the 8085AH type bus feature of the MCS®-48.

### 13.2.1 Read/Write Cycle

All address and data is transferred over the 8 lines of BUS. As shown in Figure 13-5, a read or write cycle occurs as follows:

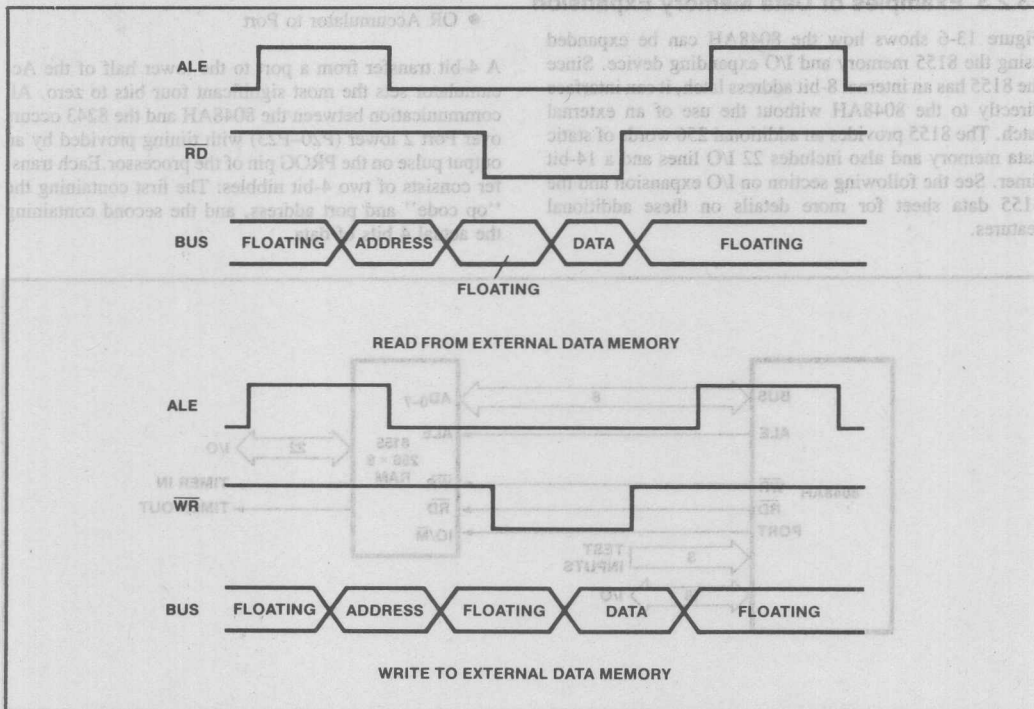


Figure 13-5. External Data Memory Timings



- 1) The contents of register R0 or R1 is outputted on BUS.
- 2) Address Latch Enable (ALE) indicates address is valid. The trailing edge of ALE is used to latch the address externally.
- 3) A read ( $\overline{RD}$ ) or write ( $\overline{WR}$ ) pulse on the corresponding output pins of the 8048AH indicates the type of data memory access in progress. Output data is valid at the trailing edge of  $\overline{WR}$  and input data must be valid at the trailing edge of  $\overline{RD}$ .
- 4) Dat (8 bits) is transferred in or out over BUS.

### 13.2.2 Addressing External Data Memory

External Data Memory is accessed with its own two-cycle move instructions.  $MOVXA, @R$  and  $MOVX@R, A$ , which transfer 8 bits of data between the accumulator and the external memory location addressed by the contents of one of the RAM Pointer Registers R0 and R1. This allows 256 locations to be addressed in addition to the resident locations. Additional pages may be added by "bank switching" with extra output lines of the 8048AH.

### 13.2.3 Examples of Data Memory Expansion

Figure 13-6 shows how the 8048AH can be expanded using the 8155 memory and I/O expanding device. Since the 8155 has an internal 8-bit address latch, it can interface directly to the 8048AH without the use of an external latch. The 8155 provides an additional 256 words of static data memory and also includes 22 I/O lines and a 14-bit timer. See the following section on I/O expansion and the 8155 data sheet for more details on these additional features.

## 13.3 EXPANSION OF INPUT/OUTPUT

There are four possible modes of I/O expansion with the 8048AH: one using a special low-cost expander, the 8243; another using standard MCS-80/85 I/O devices; and a third using the combination memory I/O expander devices the 8155, 8355, and 8755. It is also possible to expand using standard TTL devices as shown in Chapter 5.

### 13.3.1 I/O Expander Device

The most efficient means of I/O expansion for small systems is the 8243 I/O Expander Device which requires only 4 port lines (lower half of Port 2) for communication with the 8048AH. The 8243 contains four 4-bit I/O ports which serve as an extension of the on-chip I/O and are addressed as ports #4-7 (see Figure 13-7). The following operations may be performed on these ports:

- Transfer Accumulator to Port
- Transfer Port to Accumulator
- AND Accumulator to Port
- OR Accumulator to Port

A 4-bit transfer from a port to the lower half of the Accumulator sets the most significant four bits to zero. All communication between the 8048AH and the 8243 occurs over Port 2 lower (P20-P23) with timing provided by an output pulse on the PROG pin of the processor. Each transfer consists of two 4-bit nibbles: The first containing the "op code" and port address, and the second containing the actual 4 bits of data.

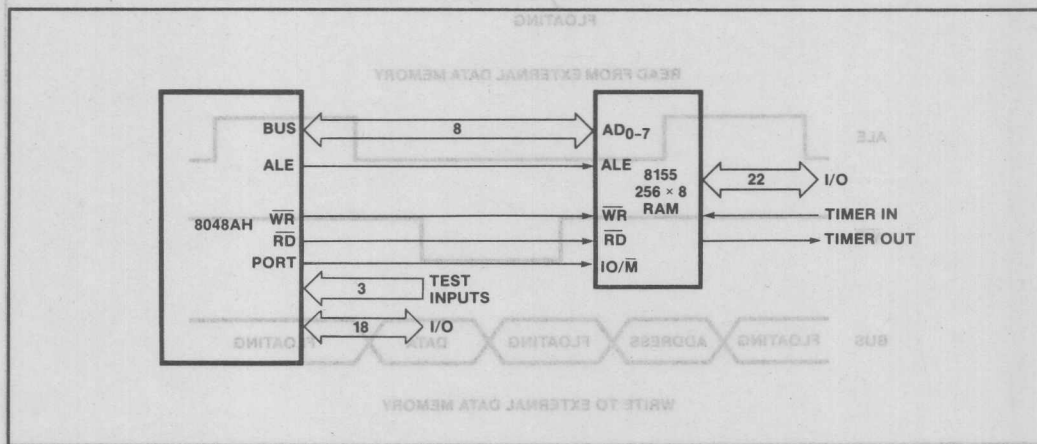


Figure 13-6. 8048AH Interface to 256 x 8 Standard Memories

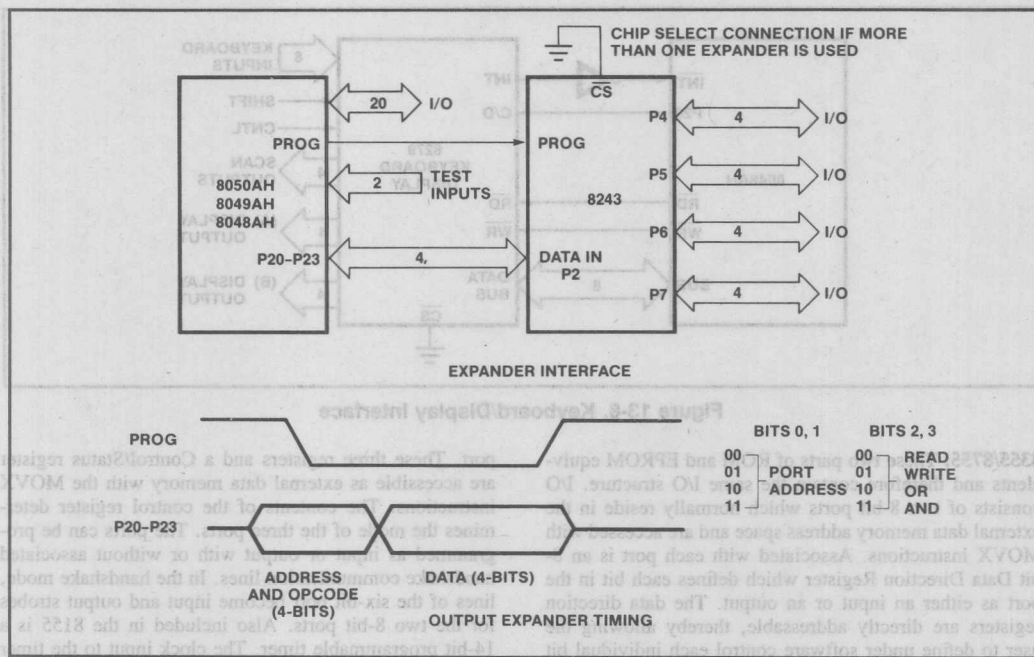
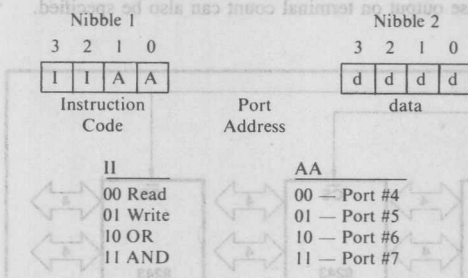


Figure 13-7. 8243 Expander I/O Interface



A high to low transition of the PROG line indicates that address is present, while a low to high transition indicates the presence of data. Additional 8243's may be added to the four-bit bus and chip selected using additional output lines from the 8048AH/8748H.

#### I/O PORT CHARACTERISTICS

Each of the four 4-bit ports of the 8243 can serve as either input or output and can provide high drive capability in both the high and low state.

#### 13.3.2 I/O Expansion with Standard Peripherals

Standard MCS-80/85 type I/O devices may be added to the MCS®-48 using the same bus and timing used for Data Memory expansion. Figure 13-8 shows an example of how an 8048AH can be connected to an MCS-85 peripheral. I/O devices reside on the Data Memory bus and in the data memory address space and are accessed with the same MOVX instructions. (See the previous section on data memory expansion for a description of timing.) The following are a few of the Standard MCS-80 devices which are very useful in MCS®-48 systems:

- 8214 Priority Interrupt Encoder
- 8251 Serial Communications Interface
- 8255 General Purpose Programmable I/O
- 8279 Keyboard/Display Interface
- 8253 Interval Timer

#### 13.3.3 Combination Memory and I/O Expanders

As mentioned in the sections on program and data memory expansion, the 8355/8755 and 8155 expanders also contain I/O capability.

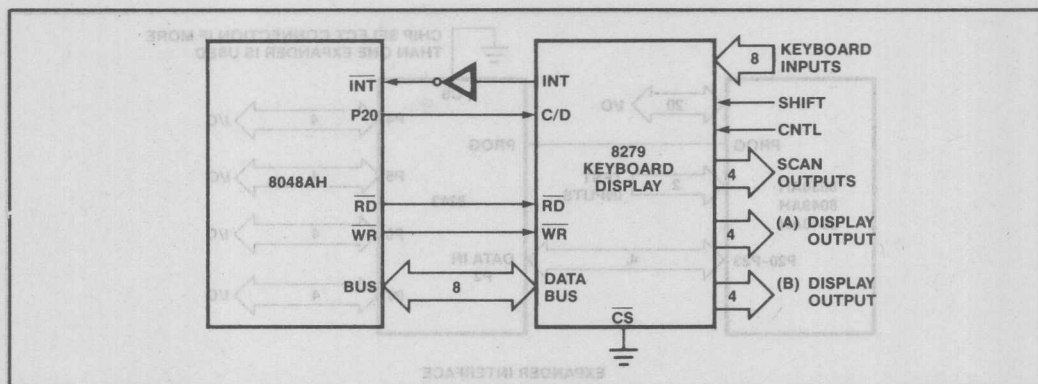


Figure 13-8. Keyboard/Display Interface

**8355/8755:** These two parts of ROM and EPROM equivalents and therefore contain the same I/O structure. I/O consists of two 8-bit ports which normally reside in the external data memory address space and are accessed with MOVX instructions. Associated with each port is an 8-bit Data Direction Register which defines each bit in the port as either an input or an output. The data direction registers are directly addressable, thereby allowing the user to define under software control each individual bit of the ports as either input or output. All outputs are statically latched and double buffered. Inputs are not latched.

**8155/8156:** I/O on the 8155/8156 is configured as two 8-bit programmable I/O ports and one 6-bit programmable

port. These three registers and a Control/Status register are accessible as external data memory with the MOVX instructions. The contents of the control register determines the mode of the three ports. The ports can be programmed as input or output with or without associated handshake communication lines. In the handshake mode, lines of the six-bit port become input and output strobes for the two 8-bit ports. Also included in the 8155 is a 14-bit programmable timer. The clock input to the timer and the timer overflow output are available on external pins. The timer can be programmed to stop on terminal count or to continuously reload itself. A square wave or pulse output on terminal count can also be specified.

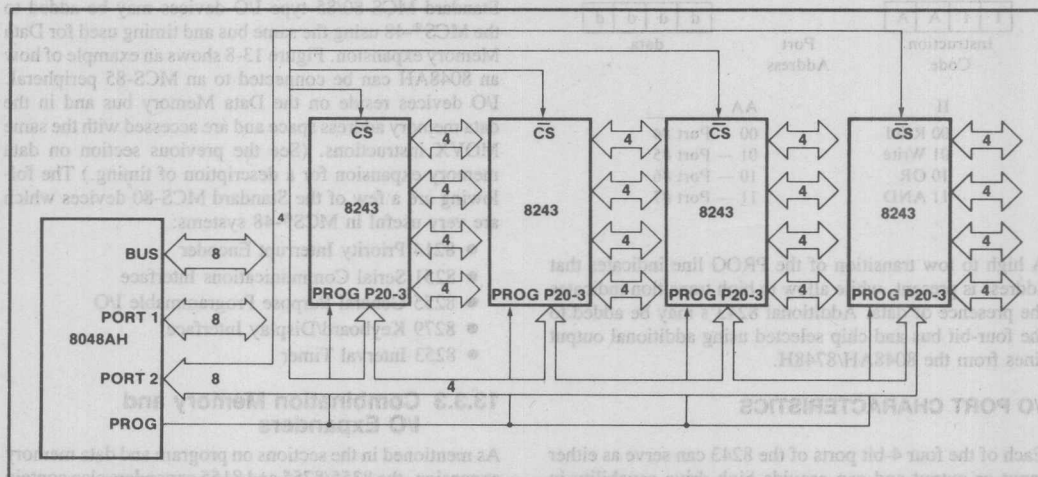


Figure 13-9. Low Cost I/O Expansion

## I/O EXPANSION EXAMPLES

Figure 13-9 shows the expansion of I/O using multiple 8243's. The only difference from a single 8243 system is the addition of chip selects provided by additional 8048AH output lines. Two output lines and a decoder could also be used to address the four chips. Large numbers of 8243's would require a chip select decoder chip such as the 8205 to save I/O pins.

Figure 13-10 shows the 8048AH interface to a standard MCS®-80 peripheral; in this case, the 8255 Programmable Peripheral Interface, a 40-pin part which provides three 8-bit programmable I/O ports. The 8255 bus interface is typical of programmable MCS®-80 peripherals with an 8-bit bidirectional data bus, a RD and WR input for Read/Write control, a CS (chip select) input used to enable the Read/Write control logic and the address inputs used to select various internal registers.

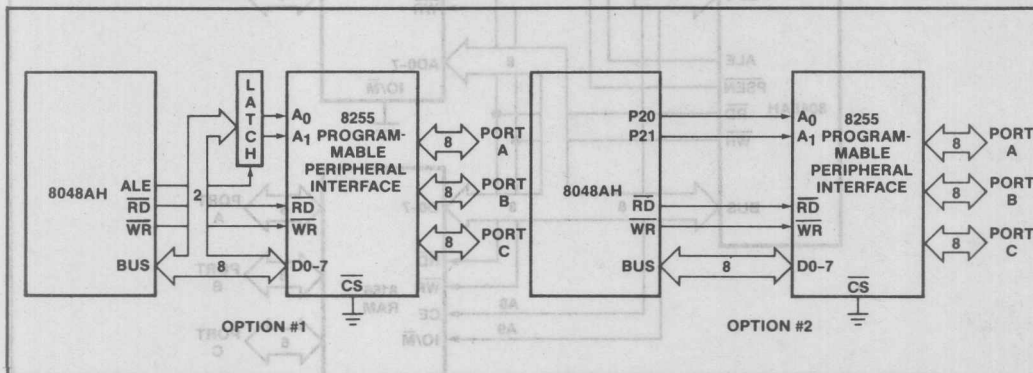


Figure 13-10. Interface to MCS®-80 Peripherals

Interconnection to the 8048AH is very straightforward with BUS, RD, and WR connecting directly to the corresponding pins on the 8255. The only design consideration is the way in which the internal registers of the 8255 are to be addressed. If the registers are to be addressed as external data memory using the MOVX instructions, the appropriate number of address bits (in this case, 2) must be latched on BUS using ALE as described in the section on external data memories. If only a single device is connected to BUS, the 8255 may be continuously selected by grounding  $\overline{CS}$ . If multiple 8255's are used, additional address bits can be latched and used as chip selects.

A second addressing method eliminates external latches and chip select decoders by using output port lines as address and chip select lines directly. This method, of course, requires the setting of an output port with address information prior to executing a MOVX instruction.

### 13.4 MULTI-CHIP MCS®-48 SYSTEMS

Figure 13-11 shows the addition of two memory expanders to the 8048AH, one 8355/8755 ROM and one 8156 RAM. The main consideration in designing such a system is the

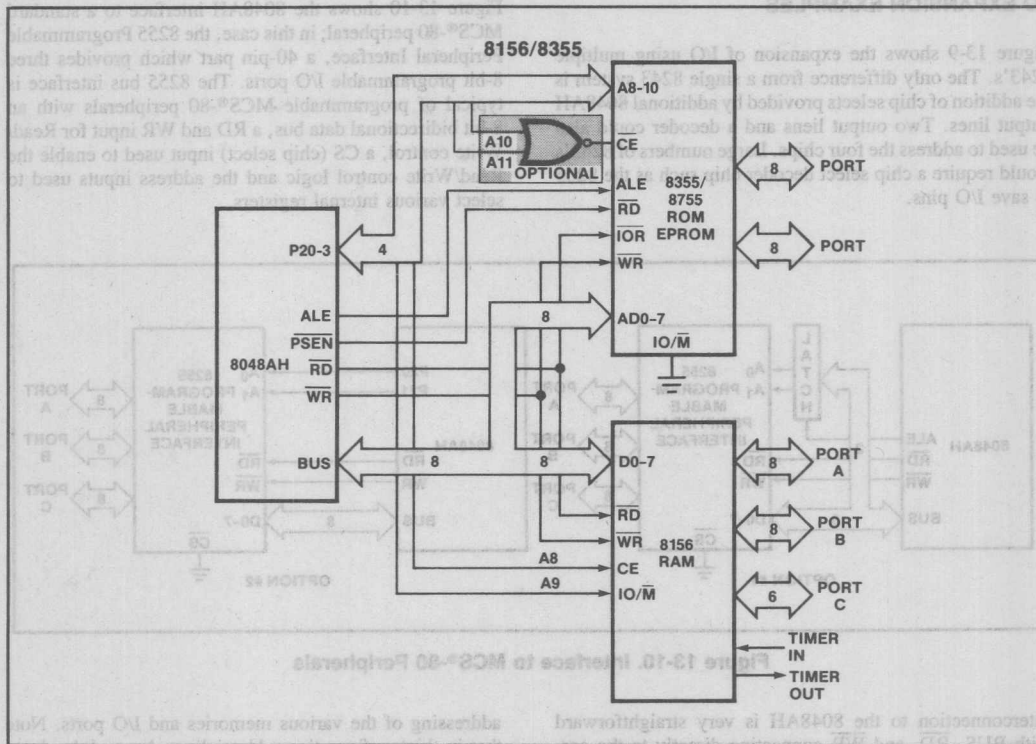
addressing of the various memories and I/O ports. Note that in this configuration address lines  $A_{10}$  and  $A_{11}$  have been tied to chip select the 8355. This ensures that the chip is active for all external program memory fetches in the 1K to 3K range and is disabled for all other addresses. This gating has been added to allow the I/O port of the 8355 to be used. If the chip was left selected all the time, there would be conflict between these ports and the RAM and I/O of the 8156. The NOR gate could be eliminated and  $A_{11}$  connected directly to the CE (instead of  $\overline{CE}$ ) input of the 8355; however, this would create a 1K word "hole" in the program memory by causing the 8355 to be active in the 2K and 4K range instead of the normal 1K to 3K range.

In this system the various locations are addressed as follows:

- Data RAM — Addresses 0 to 255 when Port 2 Bit 0 has been previously set = 1 and Bit 1 set = 0
- RAM I/O — Addresses 0 to 3 when Port 2 Bit 0 = 1 and Bit 1 = 1
- ROM I/O — Addresses 0 to 3 when Port 2 Bit 2 or Bit 3 = 1

See the memory map in Figure 13-12.





**Figure 13-11. The Three-Component MCS®-48 System**

### 13.5 MEMORY BANK SWITCHING

Certain systems may require more than the 4K words of program memory which are directly addressable by the program counter or more than the 256 data memory and I/O locations directly addressable by the pointer registers R0 and R1. These systems can be achieved using "bank switching" techniques. Bank switching is merely the selection of various blocks of "banks" of memory using dedicated output port lines from the processor. In the case of the 8048AH, program memory is selected in blocks of 4K words at a time, while data memory and I/O are enabled 256 words at a time.

The most important consideration in implementing two or more banks is the software required to cross the bank boundaries. Each crossing of the boundary requires that the processor first write a control bit to an output port before accessing memory or I/O in the new bank. If program memory is being switched, programs should be organized to keep boundary crossings to a minimum.

Jumping to subroutines across the boundary should be avoided when possible since the programmer must keep track of which bank to return to after completion of the subroutine. If these subroutines are to be nested and accessed from either bank, a software "stack" should be implemented to save the bank switch bit just as if it were another bit of the program counter.

From a hardware standpoint bank switching is very straightforward and involves only the connection of an I/O line or lines as bank enable signals. These enables are ANDed with normal memory and I/O chip select signals to activate the proper bank.

### 13.6 CONTROL SIGNAL SUMMARY

Table 13 summarizes the instructions which activate the various control outputs of the MCS<sup>®</sup>-48 processors. During all other instructions these outputs are driven to the active state.

Table 13-1. MCS®-48 Control Signals

| Control Signal | When Active   |
|----------------|---|
| RD             | During MOVX, A, @R or INS Bus   |
| WR             | During MOVX @R, A or OUTL Bus   |
| ALE            | Every Machine Cycle   |
| PSEN           | During Fetch of external program memory (instruction or immediate data) |
| PROG           | During MOVD, A,P ANLD P,A MOVD P,A ORLD P,A                             |

### 13.7 PORT CHARACTERISTICS

#### 13.7 BUS Port Operations

The BUS port can operate in three different modes: as a latched I/O port, as a bidirectional bus port, or as a program memory address output when external memory is used. The BUS port lines are either active high, active low, or high impedance (floating).

The latched mode (INS, OUTL) is intended for use in the single-chip configuration where BUS is not begin used as an expander port. OUTL and MOVX instructions can be mixed if necessary. However, a previously latched output will be destroyed by executing a MOVX instruction and BUS will be left in the high impedance state. INS does not put the BUS in a high impedance state. Therefore, the use of MOVX after OUTL to put the BUS in a high impedance state is necessary before an INS instruction intended to read an external word (as opposed to the previously latched value).

OUTL should never be used in a system with external program memory, since latching BUS can cause the next instruction, if external, to be fetched improperly.

#### 13.7.2 Port 2 Operations

The lower half of Port 2 can be used in three different ways: as a quasi-bidirectional static port, as an 8243 expander port, and to address external program memory.

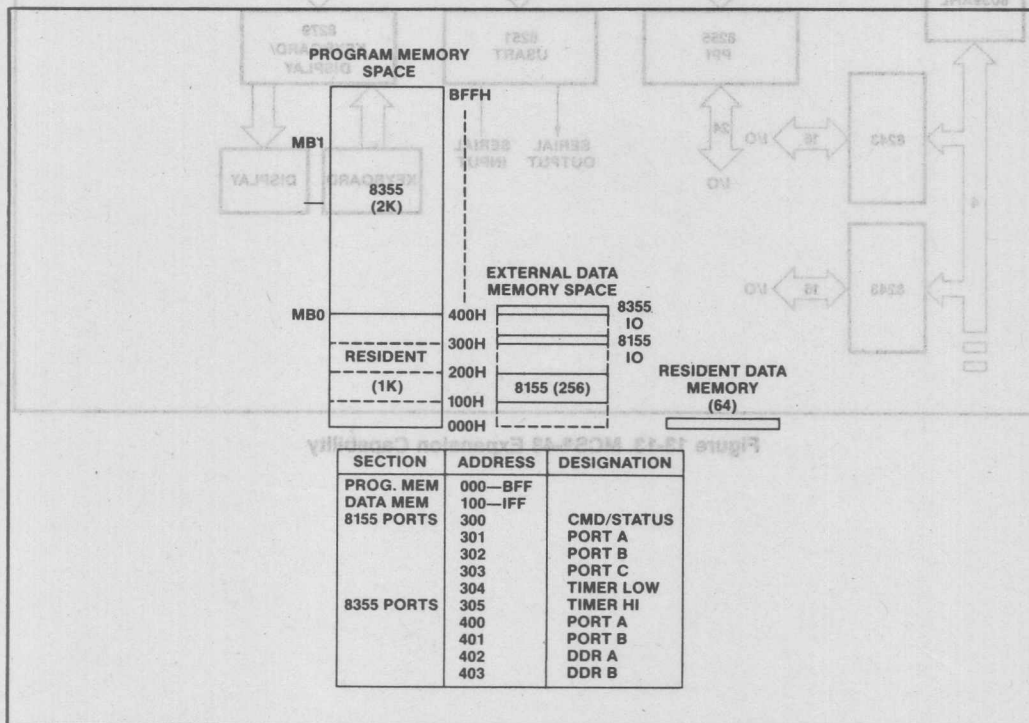


Figure 13-12. Memory Map for Three-Component MCS®-48 Family

In all cases outputs are driven low by an active device and driven high momentarily by a low impedance device and held high by a high impedance device to VCC.

The port may contain latched I/O data prior to its use in another mode without affecting operation of either. If lower Port 2 (P20-3) is used to output address for an external program memory fetch, the I/O information pre-

viously latched will be automatically removed temporarily while address is present, then retored when the fetch is complete. However, if lower Port 2 is used to communicate with an 8243, previously latched I/O information will be removed and not restored. After an input from the 8243, P20-3 will be left in the input mode (floating). After an output to the 8243, P20-3 will contain the value written, ANDed, or ORed to the 8243 port.

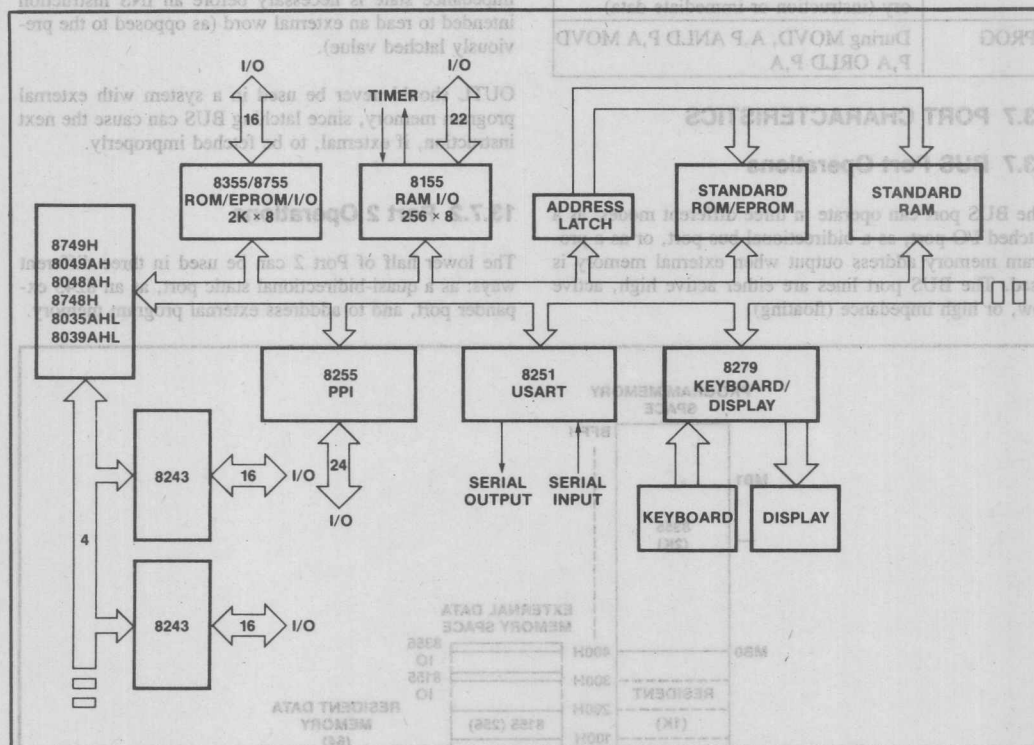
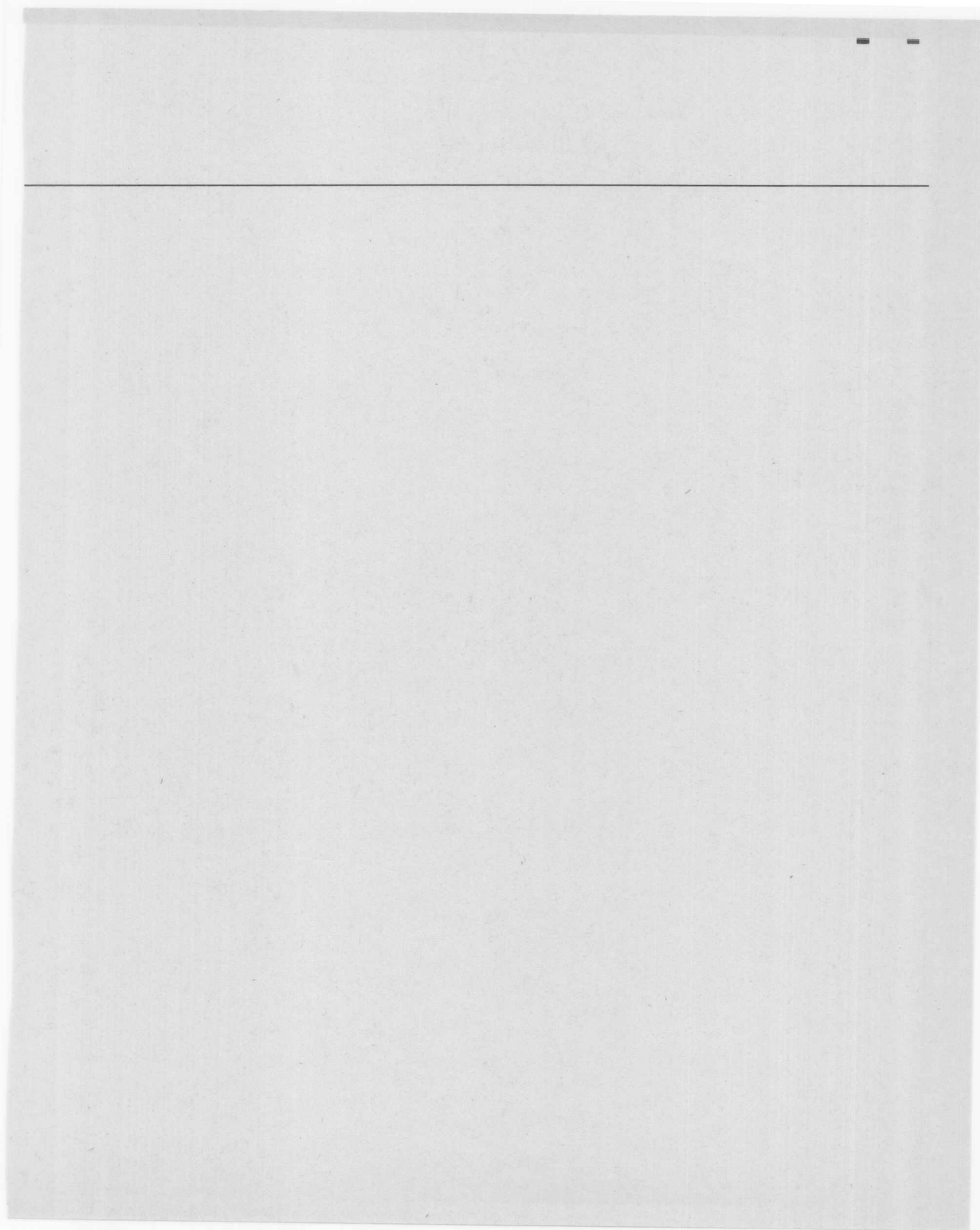


Figure 13-13. MCS-48 Expansion Capability

| ADDRESS | DESIGNATION |
|---------|-------------|
| 000-00F | PROD. MEM.  |
| 100-10F | DATA MEM.   |
| 200-20F | PORTS       |
| 300-30F | PORT A      |
| 400-40F | PORT B      |
| 500-50F | PORT C      |
| 600-60F | TIMER LOW   |
| 700-70F | TIMER HI    |
| 800-80F | PORT A      |
| 900-90F | PORT B      |
| A00-A0F | PORT C      |
| B00-B0F | PORT A      |
| C00-C0F | PORT B      |







# CHAPTER 14

## MCS®-48 INSTRUCTION SET

### 14.0 INTRODUCTION

The MCS®-48 instruction set is extensive for a machine of its size and has been tailored to be straightforward and very efficient in its use of program memory. All instructions are either one or two bytes in length and over 80% are only one byte long. Also, all instructions execute in either one or two cycles and over 50% of all instructions execute in a single cycle. Double cycle instructions include all immediate instructions, and all I/O instructions.

The MCS-48 microcomputers have been designed to handle arithmetic operations efficiently in both binary and BCD as well as handle the single-bit operations required in control applications. Special instructions have also been included to simplify loop counters, table look-up routines, and N-way branch routines.

### 14.0.1 Data Transfers

As can be seen in Figure 14.1, the 8-bit accumulator is the central point for all data transfers within the 8048. Data can be transferred between the 8 registers of each working register bank and the accumulator directly, i.e., the source or destination register is specified by the instruction. The remaining locations of the internal RAM array are referred to as Data Memory and are addressed indirectly via an address stored in either R0 or R1 of the active register bank. R0 and R1 are also used to indirectly address external data memory when it is present. Transfers to and from internal RAM require one cycle, while transfers to external RAM require two. Constants stored in Program Memory can be loaded directly to the accumulator and to the 8 working registers. Data can also be transferred directly between the accumulator and the on-

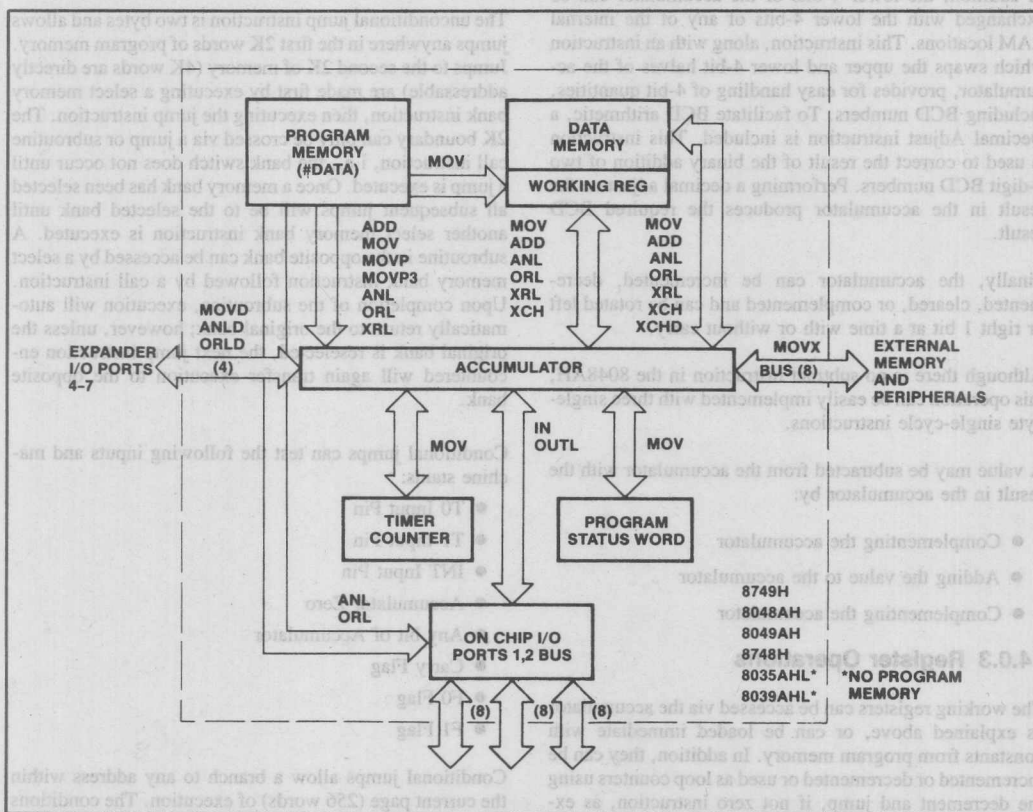


Figure 14-1: Data Transfer Instructions

board timer counter or the accumulator and the Program Status word (PSW). Writing to the PSW alters machine status accordingly and provides a means of restoring status after an interrupt or of altering the stack pointer if necessary.

#### 14.0.2 Accumulator Operations

Immediate data, data memory, or the working registers can be added with or without carry to the accumulator. These sources can also be ANDed, ORed, or Exclusive ORed to the accumulator. Data may be moved to or from the accumulator and working registers or data memory. The two values can also be exchanged in a single operation.

In addition, the lower 4 bits of the accumulator can be exchanged with the lower 4-bits of any of the internal RAM locations. This instruction, along with an instruction which swaps the upper and lower 4-bit halves of the accumulator, provides for easy handling of 4-bit quantities, including BCD numbers. To facilitate BCD arithmetic, a Decimal Adjust instruction is included. This instruction is used to correct the result of the binary addition of two 2-digit BCD numbers. Performing a decimal adjust on the result in the accumulator produces the required BCD result.

Finally, the accumulator can be incremented, decremented, cleared, or complemented and can be rotated left or right 1 bit at a time with or without carry.

Although there is no subtract instruction in the 8048AH, this operation can be easily implemented with three single-byte single-cycle instructions.

A value may be subtracted from the accumulator with the result in the accumulator by:

- Complementing the accumulator
- Adding the value to the accumulator
- Complementing the accumulator

#### 14.0.3 Register Operations

The working registers can be accessed via the accumulator as explained above, or can be loaded immediate with constants from program memory. In addition, they can be incremented or decremented or used as loop counters using the decrement and jump, if not zero instruction, as explained under branch instructions.

All Data Memory including working registers can be accessed with indirect instructions via R0 and R1 and can be incremented.

#### 14.0.4 Flags

There are four user-accessible flags in the 8048AH: Carry, Auxiliary Carry, F0 and F1. Carry indicates overflow of the accumulator, and Auxiliary Carry is used to indicate overflow between BCD digits and is used during decimal-adjust operation. Both Carry and Auxiliary Carry are accessible as part of the program status word and are stored on the stack during subroutines. F0 and F1 are undedicated general-purpose flags to be used as the programmer desires. Both flags can be cleared or complemented and tested by conditional jump instructions. F0 is also accessible via the Program Status word and is stored on the stack with the carry flags.

#### 14.0.5 Branch Instructions

The unconditional jump instruction is two bytes and allows jumps anywhere in the first 2K words of program memory. Jumps to the second 2K of memory (4K words are directly addressable) are made first by executing a select memory bank instruction, then executing the jump instruction. The 2K boundary can only be crossed via a jump or subroutine call instruction, i.e., the bank switch does not occur until a jump is executed. Once a memory bank has been selected all subsequent jumps will be to the selected bank until another select memory bank instruction is executed. A subroutine in the opposite bank can be accessed by a select memory bank instruction followed by a call instruction. Upon completion of the subroutine, execution will automatically return to the original bank; however, unless the original bank is reselected, the next jump instruction encountered will again transfer execution to the opposite bank.

Conditional jumps can test the following inputs and machine status:

- T0 Input Pin
- T1 Input Pin
- $\overline{\text{INT}}$  Input Pin
- Accumulator Zero
- Any bit of Accumulator
- Carry Flag
- F0 Flag
- F1 Flag

Conditional jumps allow a branch to any address within the current page (256 words) of execution. The conditions tested are the instantaneous values at the time the conditional jump is executed. For instance, the jump on accumulator zero instruction tests the accumulator itself, not an intermediate zero flag.

The decrement register and jump if not zero instruction combines a decrement and a branch instruction to create an instruction very useful in implementing a loop counter. This instruction can designate any one of the 8 working registers as a counter and can effect a branch to any address within the current page of execution.

A single-byte indirect jump instruction allows the program to be vectored to any one of several different locations based on the contents of the accumulator. The contents of the accumulator points to a location in program memory which contains the jump address. The 8-bit jump address refers to the current page of execution. This instruction could be used, for instance, to vector to any one of several routines based on an ASCII character which has been loaded in the accumulator. In this way ASCII key inputs can be used to initiate various routines.

#### 14.0.6 Subroutines

Subroutines are entered by executing a call instruction. Calls can be made like unconditional jumps to any address in a 2K word bank, and jumps across the 2K boundary are executed in the same manner. Two separate return instructions determine whether or not status (upper 4-bits of PSW) is restored upon return from the subroutine.

The return and restore status instruction also signals the end of an interrupt service routine if one has been in progress.

#### 14.0.7 Timer Instructions

The 8-bit on board timer/counter can be loaded or read via the accumulator while the counter is stopped or while counting. The counter can be started as a timer with an internal clock source or an event counter or timer with an external clock applied to the T1 input pin. The instruction executed determines which clock source is used. A single instruction stops the counter whether it is operating with an internal or an external clock source. In addition, two instructions allow the timer interrupt to be enabled or disabled.

#### 14.0.8 Control Instructions

Two instructions allow the external interrupt source to be enabled or disabled. Interrupts are initially disabled and are automatically disabled while an interrupt service routine is in progress and re-enabled afterward.

There are four memory bank select instructions, two to designate the active working register bank and two to control program memory banks. The operation of the program memory bank switch is explained in section 13.1.2.

The working register bank switch instructions allow the programmer to immediately substitute a second 8-register working register bank for the one in use. This effectively provides 16 working registers or it can be used as a means of quickly saving the contents of the registers in response to an interrupt. The user has the option to switch or not to switch banks on interrupt. However, if the banks are switched, the original bank will be automatically restored upon execution of a return and restore status instruction at the end of the interrupt service routine.

A special instruction enables an internal clock, which is the XTAL frequency divided by three to be output on pin T0. This clock can be used as a general-purpose clock in the user's system. This instruction should be used only to initialize the system since the clock output can be disabled only by application of system reset.

#### 14.0.9. Input/Output Instructions

Ports 1 and 2 are 8-bit static I/O ports which can be loaded to and from the accumulator. Outputs are statically latched but inputs are not latched and must be read while inputs are present. In addition, immediate data from program memory can be ANDed or ORed directly to Port 1 and Port 2 with the result remaining on the port. This allows "masks" stored in program memory to selectively set or reset individual bits of the I/O ports. Ports 1 and 2 are configured to allow input on a given pin by first writing a "1" out to the pin.

An 8-bit port called BUS can also be accessed via the accumulator and can have statically latched outputs as well. It too can have immediate data ANDed or ORed directly to its outputs, however, unlike ports 1 and 2, all eight lines of BUS must be treated as either input or output at any one time. In addition to being a static port, BUS can be used as a true synchronous bi-directional port using the Move External instructions used to access external data memory. When these instructions are executed, a corresponding READ or WRITE pulse is generated and data is valid only at that time. When data is not being transferred, BUS is in a high impedance state. Note that the OUTL, ANL, and the ORL instructions for the BUS are for use with internal program memory only.

The basic three on-board I/O ports can be expanded via a 4-bit expander bus using half of port 2. I/O expander devices on this bus consist of four 4-bit ports which are addressed as ports 4 through 7. These ports have their own AND and OR instructions like the on-board ports as well as move instructions to transfer data in or out. The expander AND and OR instructions, however, combine the contents of accumulator with the selected port rather than immediate data as is done with the on-board ports.



I/O devices can also be added externally using the BUS port as the expansion bus. In this case the I/O ports become "memory mapped", i.e., they are addressed in the same way as external data memory and exist in the external data memory address space addressed by pointer register R0 or R1.

## 14.1 INSTRUCTION SET DESCRIPTION

The following pages describe the MCS<sup>®</sup>-48 instruction set in detail. The instruction set is first summarized with instructions grouped functionally. This summary page is followed by a detailed description listed alphabetically by mnemonic opcode.

The alphabetical listing includes the following information.

- Mnemonic
- Machine Code
- Verbal Description
- Symbolic Description
- Assembly Language Example

The machine code is represented with the most significant bit (7) to the left and two byte instructions are represented with the first byte on the left. The assembly language examples are formulated as follows:

Arbitrary

Label: Mnemonic, Operand;

Descriptive Comment

Ports 1 and 2 are 8-bit static I/O ports which can be loaded to and from the accumulator. Outputs are statically latched but inputs are not latched and must be read while inputs are present. In addition, immediate data from program memory can be ANDed or ORed directly to Port 1 and Port 2 with the result remaining on the port. This allows "mask" stored in program memory to selectively set or reset individual bits of the I/O ports. Ports 1 and 2 are configured to allow input on a given pin by first writing a "1" out to the pin.

An 8-bit port called BUS can also be accessed via the accumulator and can have statically latched outputs as well. It too can have immediate data ANDed or ORed directly to its outputs, however, unlike ports 1 and 2, all eight lines of BUS must be treated as either input or output at any one time. In addition to being a static port, BUS can be used as a true synchronous bi-directional port using the Move External instructions used to access external data memory. When these instructions are executed, a corresponding READ or WRITE pulse is generated and data is valid only at that time. When data is not being transferred, BUS is in a high impedance state. Note that the OUT, ANI, and the ORL instructions for the BUS are for use with internal program memory only.

The basic three on-board I/O ports can be expanded via a 4-bit expander bus using half of port 2. I/O expander devices on this bus consist of four 4-bit ports which are addressed as ports 4 through 7. These ports have their own AND and OR instructions like the on-board ports as well as move instructions to transfer data in or out. The expander AND and OR instructions, however, combine the contents of accumulator with the selected port rather than immediate data as is done with the on-board ports.

Subroutines are entered by executing a call instruction. Calls can be made like unconditional jumps to any address in a 2K word bank, and jumps across the 2K boundary are executed in the same manner. Two separate return instructions determine whether or not status (upper 4-bits of PSW) is restored upon return from the subroutine.

The return and restore status instruction also signals the end of an interrupt service routine if one has been in progress.

## 14.0.7 Timer Instructions

The 8-bit on-board timer/counter can be loaded or read via the accumulator while the counter is stopped or while counting. The counter can be started as a timer with an internal clock source or an event counter or timer with an external clock applied to the TI input pin. The instruction executed determines which clock source is used. A single instruction stops the counter whether it is operating with an internal or an external clock source. In addition, two instructions allow the timer interrupt to be enabled or disabled.

## 14.0.8 Control Instructions

Two instructions allow the external interrupt source to be enabled or disabled. Interrupts are initially disabled and are automatically disabled while an interrupt service routine is in progress and re-enabled afterward.

There are four memory bank select instructions, two to designate the active working register bank and two to control program memory banks. The operation of the program memory bank switch is explained in section 13.1.2.

# Instruction Set Summary

| Mnemonic            | Description                   | Bytes | Cycle |
|---------------------|-------------------------------|-------|-------|
| <b>Accumulator</b>  |                               |       |       |
| ADD A, R            | Add register to A             | 1     | 1     |
| ADD A, @R           | Add data memory to A          | 1     | 1     |
| ADD A, # data       | Add immediate to A            | 2     | 2     |
| ADDC A, R           | Add register with carry       | 1     | 1     |
| ADDC A, @R          | Add data memory with carry    | 1     | 1     |
| ADDC A, # data      | Add immediate with carry      | 2     | 2     |
| ANL A, R            | And register to A             | 1     | 1     |
| ANL A, @R           | And data memory to A          | 1     | 1     |
| ANL A, # data       | And immediate to A            | 2     | 2     |
| ORL A, R            | Or register to A              | 1     | 1     |
| ORL A, @R           | Or data memory to A           | 1     | 1     |
| ORL A, # data       | Or immediate to A             | 2     | 2     |
| XRL A, R            | Exclusive Or register to A    | 1     | 1     |
| XRL A, @R           | Exclusive or data memory to A | 1     | 1     |
| XRL A, # data       | Exclusive or immediate to A   | 2     | 2     |
| INC A               | Increment A                   | 1     | 1     |
| DEC A               | Decrement A                   | 1     | 1     |
| CLR A               | Clear A                       | 1     | 1     |
| CPL A               | Complement A                  | 1     | 1     |
| DA A                | Decimal adjust A              | 1     | 1     |
| SWAP A              | Swap nibbles of A             | 1     | 1     |
| RL A                | Rotate A left                 | 1     | 1     |
| RLC A               | Rotate A left through carry   | 1     | 1     |
| RR A                | Rotate A right                | 1     | 1     |
| RRC A               | Rotate A right through carry  | 1     | 1     |
| <b>Input/Output</b> |                               |       |       |
| IN A, P             | Input port to A               | 1     | 2     |
| OUTL P, A           | Output A to port              | 1     | 2     |
| ANL P, # data       | And immediate to port         | 2     | 2     |
| ORL P, # data       | Or immediate to port          | 2     | 2     |
| *INS A, BUS         | Input BUS to A                | 1     | 2     |
| *OUTL BUS, A        | Output A to BUS               | 1     | 2     |
| *ANL BUS, # data    | And immediate to BUS          | 2     | 2     |
| *ORL BUS, # data    | Or immediate to BUS           | 2     | 2     |
| MOVD A, P           | Input Expander port to A      | 1     | 2     |
| MOVD P, A           | Output A to Expander port     | 1     | 2     |
| ANLD P, A           | And A to Expander port        | 1     | 2     |
| ORLD P, A           | Or A to Expander port         | 1     | 2     |

| Mnemonic          | Description                   | Bytes | Cycles |
|-------------------|-------------------------------|-------|--------|
| <b>Registers</b>  |                               |       |        |
| INC R             | Increment register            | 1     | 1      |
| INC @R            | Increment data memory         | 1     | 1      |
| DEC R             | Decrement register            | 1     | 1      |
| <b>Branch</b>     |                               |       |        |
| JMP addr          | Jump unconditional            | 2     | 2      |
| JMPP @A           | Jump indirect                 | 1     | 2      |
| DJNZ R, addr      | Decrement register and jump   | 2     | 2      |
| JC addr           | Jump on carry = 1             | 2     | 2      |
| JNC addr          | Jump on carry = 0             | 2     | 2      |
| JZ addr           | Jump on A Zero                | 2     | 2      |
| JNZ addr          | Jump on A not Zero            | 2     | 2      |
| JT0 addr          | Jump on T0 = 1                | 2     | 2      |
| JNT0 addr         | Jump on T0 = 0                | 2     | 2      |
| JT1 addr          | Jump on T1 = 1                | 2     | 2      |
| JNT1 addr         | Jump on T1 = 0                | 2     | 2      |
| JF0 addr          | Jump on F0 = 1                | 2     | 2      |
| JF1 addr          | Jump on F1 = 1                | 2     | 2      |
| JTF addr          | Jump on timer flag = 1        | 2     | 2      |
| JNI addr          | Jump on INT = 0               | 2     | 2      |
| JBb addr          | Jump on Accumulator Bit       | 2     | 2      |
| <b>Subroutine</b> |                               |       |        |
| CALL addr         | Jump to subroutine            | 2     | 2      |
| RET               | Return                        | 1     | 2      |
| RETR              | Return and restore status     | 1     | 2      |
| <b>Flags</b>      |                               |       |        |
| CLR C             | Clear Carry                   | 1     | 1      |
| CPL C             | Complement Carry              | 1     | 1      |
| CLR F0            | Clear Flag 0                  | 1     | 1      |
| CPL F0            | Complement Flag 0             | 1     | 1      |
| CLR F1            | Clear Flag 1                  | 1     | 1      |
| CPL F1            | Complement Flag 1             | 1     | 1      |
| <b>Data Moves</b> |                               |       |        |
| MOV A, R          | Move register to A            | 1     | 1      |
| MOV A, @R         | Move data memory to A         | 1     | 1      |
| MOV A, # data     | Move immediate to A           | 2     | 2      |
| MOV R, A          | Move A to register            | 1     | 1      |
| MOV @R, A         | Move A to data memory         | 1     | 1      |
| MOV R, # data     | Move immediate to register    | 2     | 2      |
| MOV @R, # data    | Move immediate to data memory | 2     | 2      |
| MOV A, PSW        | Move PSW to A                 | 1     | 1      |
| MOV PSW, A        | Move A to PSW                 | 1     | 1      |

Mnemonics copyright Intel Corporation 1983.

\*For use with internal memory only.

# MCS®-48 INSTRUCTION SET

8048AH/8748H/8049AH/8050AH/8749H

## Instruction Set Summary (Con't)

| Mnemonic                   | Description                       | Bytes | Cycle |
|----------------------------|-----------------------------------|-------|-------|
| <b>Data Moves (Cont'd)</b> |                                   |       |       |
| <b>XCH A, R</b>            | Exchange A and register           | 1     | 1     |
| <b>XCH A, @R</b>           | Exchange A and data memory        | 1     | 1     |
| <b>XCHD A, @R</b>          | Exchange nibble of A and register | 1     | 1     |
| <b>MOVX A, @R</b>          | Move external data memory to A    | 1     | 2     |
| <b>MOVX @R, A</b>          | Move A to external data memory    | 1     | 2     |
| <b>MOVP A, @A</b>          | Move to A from current page       | 1     | 2     |
| <b>MOVP3 A, @A</b>         | Move to A from Page 3             | 1     | 2     |
| <b>Timer/Counter</b>       |                                   |       |       |
| <b>MOV A, T</b>            | Read Timer/Counter                | 1     | 1     |
| <b>MOV T, A</b>            | Load Timer/Counter                | 1     | 1     |
| <b>STRT T</b>              | Start Timer                       | 1     | 1     |
| <b>STRT CNT</b>            | Start Counter                     | 1     | 1     |
| <b>STOP TCNT</b>           | Stop Timer/Counter                | 1     | 1     |
| <b>EN TCNTI</b>            | Enable Timer/Counter Interrupt    | 1     | 1     |
| <b>DIS TCNTI</b>           | Disable Timer/Counter Interrupt   | 1     | 1     |
| <b>Control</b>             |                                   |       |       |
| <b>EN I</b>                | Enable external Interrupt         | 1     | 1     |
| <b>DIS I</b>               | Disable external Interrupt        | 1     | 1     |
| <b>SEL RB0</b>             | Select register bank 0            | 1     | 1     |
| <b>SEL RB1</b>             | Select register bank 1            | 1     | 1     |
| <b>SEL MB0</b>             | Select memory bank 0              | 1     | 1     |
| <b>SEL MB1</b>             | Select memory bank 1              | 1     | 1     |
| <b>ENT0 CLK</b>            | Enable clock output on T0         | 1     | 1     |
| <b>NOP</b>                 | No Operation                      | 1     | 1     |

| Mnemonic                 | Description                | Bytes | Cycle |
|--------------------------|----------------------------|-------|-------|
| <b>Control</b>           |                            |       |       |
| <b>EN I</b>              | Enable external Interrupt  | 1     | 1     |
| <b>DIS I</b>             | Disable external Interrupt | 1     | 1     |
| <b>SEL RB0</b>           | Select register bank 0     | 1     | 1     |
| <b>SEL RB1</b>           | Select register bank 1     | 1     | 1     |
| <b>SEL MB0</b>           | Select memory bank 0       | 1     | 1     |
| <b>SEL MB1</b>           | Select memory bank 1       | 1     | 1     |
| <b>ENT0 CLK</b>          | Enable clock output on T0  | 1     | 1     |
| <b>NOP</b>               | No Operation               | 1     | 1     |
| <b>Input/Output</b>      |                            |       |       |
| <b>IN A, P</b>           | Input port to A            | 1     | 2     |
| <b>OUT A, P</b>          | Output A to port           | 1     | 2     |
| <b>IN A, P, #data</b>    | And immediate to port      | 2     | 2     |
| <b>OUT A, P, #data</b>   | Or immediate to port       | 2     | 2     |
| <b>IN A, BUS</b>         | Input BUS to A             | 1     | 2     |
| <b>OUT A, BUS</b>        | Output A to BUS            | 1     | 2     |
| <b>IN A, BUS, #data</b>  | And immediate to BUS       | 2     | 2     |
| <b>OUT A, BUS, #data</b> | Or immediate to BUS        | 2     | 2     |
| <b>IN A, P, #data</b>    | Input Expander port to A   | 1     | 2     |
| <b>OUT A, P, #data</b>   | Output A to Expander port  | 1     | 2     |
| <b>IN A, P, #data</b>    | And A to Expander port     | 1     | 2     |
| <b>OUT A, P, #data</b>   | Or A to Expander port      | 1     | 2     |

Mnemonics copyright Intel Corporation 1983.

Mnemonics copyright Intel Corporation 1983.  
For use with internal memory only.

## MCS®-48 INSTRUCTION SET

### Symbols and Abbreviations Used

|        |                                     |
|--------|-------------------------------------|
| A      | Accumulator                         |
| AC     | Auxiliary Carry                     |
| addr   | 12-Bit Program Memory Address       |
| Bb     | Bit Designator (b = 0-7)            |
| BS     | Bank Switch                         |
| BUS    | BUS Port                            |
| C      | Carry                               |
| CLK    | Clock                               |
| CNT    | Event Counter                       |
| CRR    | Conversion Result Register          |
| D      | Mnemonic for 4-Bit Digit (Nibble)   |
| data   | 8-Bit Number or Expression          |
| DBF    | Memory Bank Flip-Flop               |
| F0, F1 | Flag 0, Flag 1                      |
| I      | Interrupt                           |
| P      | Mnemonic for "in-page" Operation    |
| PC     | Program Counter                     |
| Pp     | Port Designator (p = 1, 2 or 4-7)   |
| PSW    | Program Status Word                 |
| Ri     | Data memory Pointer (i = 0, or 1)   |
| Rr     | Register Designator (r = 0-7)       |
| SP     | Stack Pointer                       |
| T      | Timer                               |
| TF     | Timer Flag                          |
| T0, T1 | Test 0, Test 1                      |
| X      | Mnemonic for External RAM           |
| #      | Immediate Data Prefix               |
| @      | Indirect Address Prefix             |
| \$     | Current Value of Program Counter    |
| (X)    | Contents of X                       |
| ((X))  | Contents of Location Addressed by X |
| ←      | Is Replaced by                      |

Mnemonics copyright Intel Corporation 1983.



**ADD A,R<sub>r</sub> Add Register Contents to Accumulator****Encoding:** 0 1 1 0 1 r r r 68H-6FH**Description:** The contents of register 'r' are added to the accumulator. Carry is affected.**Operation:**  $(A) \leftarrow (A) + (R_r)$   $r = 0-7$ **Example:** ADDR: ADD A,R6 ;ADD REG 6 CONTENTS  
;TO ACC**ADD A,@R<sub>i</sub> Add Data Memory Contents to Accumulator****Encoding:** 0 1 1 0 0 0 0 i 60H-61H**Description:** The contents of the resident data memory location addressed by register 'i' bits 0-5\*\* are added to the accumulator. Carry is affected.**Operation:**  $(A) \leftarrow (A) + ((R_i))$   $i = 0-1$ **Example:** ADDM: MOV R0, #01FH ;MOVE '1F' HEX TO REG 0  
ADD A, @R0 ;ADD VALUE OF LOCATION  
;31 TO ACC**ADD A,#data Add Immediate Data to Accumulator****Encoding:** 0 0 0 0 0 0 1 1 d<sub>7</sub> d<sub>6</sub> d<sub>5</sub> d<sub>4</sub> d<sub>3</sub> d<sub>2</sub> d<sub>1</sub> d<sub>0</sub> 03H**Description:** This is a 2-cycle instruction. The specified data is added to the accumulator. Carry is affected.**Operation:**  $(A) \leftarrow (A) + \text{data}$ **Example:** ADDID: ADD A,#ADDER ;ADD VALUE OF SYMBOL  
;ADDER TO ACC**ADDC A,R<sub>r</sub> Add Carry and Register Contents to Accumulator****Encoding:** 0 1 1 1 1 r r r 78H-7FH**Description:** The content of the carry bit is added to accumulator location 0 and the carry bit cleared. The contents of register 'r' are then added to the accumulator. Carry is affected.**Operation:**  $(A) \leftarrow (A) + (R_r) + (C)$   $r = 0-7$ **Example:** ADDRGC: ADDC A,R4 ;ADD CARRY AND REG 4  
;CONTENTS TO ACC

\*\* 0-5 in 8048AH/8748H  
 0-6 in 8049AH/8749H  
 0-7 in 8050AH

**ADDC A,@R<sub>i</sub> Add Carry and Data Memory Contents to Accumulator**

Encoding: 

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 0 | 0 | 0 | i |
|---|---|---|---|---|---|---|---|

 70H-71H

**Description:** The content of the carry bit is added to accumulator location 0 and the carry bit cleared. Then the contents of the resident data memory location addressed by register 'i' bits 0-5\*\* are added to the accumulator. Carry is affected.

**Operation:**  $(A) \leftarrow (A) + ((Ri)) + (C)$   $i = 0-1$

**Example:** ADDMC: MOV R1,#40 ;MOVE '40' DEC TO REG 1  
 ADDC A,@R1 ;ADD CARRY AND LOCATION 40  
 ;CONTENTS TO ACC

**ADDC A,@data Add Carry and Immediate Data to Accumulator**

Encoding: 

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|

|                |                |                |                |                |                |                |                |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| d <sub>7</sub> | d <sub>6</sub> | d <sub>5</sub> | d <sub>4</sub> | d <sub>3</sub> | d <sub>2</sub> | d <sub>1</sub> | d <sub>0</sub> |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|

 13H

**Description:** This is a 2-cycle instruction. The content of the carry bit is added to accumulator location 0 and the carry bit cleared. Then the specified data is added to the accumulator. Carry is affected.

**Operation:**  $(A) \leftarrow (A) + \text{data} + (C)$

**Example:** ADDC A,#225 ;ADD CARRY AND '225' DEC  
 ;TO ACC

**ANL A,R<sub>r</sub> Logical AND Accumulator with Register Mask**

Encoding: 

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 1 | r | r | r |
|---|---|---|---|---|---|---|---|

 58H-5FH

**Description:** Data in the accumulator is logically ANDed with the mask contained in working register 'r'.

**Operation:**  $(A) \leftarrow (A) \text{ AND } (Rr)$   $r = 0-7$

**Example:** ANDREG: ANL A,R3 ;AND' ACC CONTENTS WITH MASK  
 ;IN REG 3

**ANL A,@R<sub>i</sub> Logical AND Accumulator with memory Mask**

Encoding: 

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | i |
|---|---|---|---|---|---|---|---|

 50H-51H

**Description:** Data in the accumulator is logically ANDed with the mask contained in the data memory location referenced by register 'i' bits 0-5\*\*.

**Operation:**  $(A) \leftarrow (A) \text{ AND } ((Ri))$   $i = 0-1$

**Example:** ANDDM: MOV R0,#03FH ;MOVE '3F' HEX TO REG 0  
 ANL A,@R0 ;AND' ACC CONTENTS WITH  
 ;MASK IN LOCATION 63

\*\* 0-5 in 8048AH/8748H  
 0-6 in 8049AH/8749H  
 0-7 in 8050AH

**ANL A,#data Logical AND Accumulator with Immediate Mask**

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|

|                |                |                |                |                |                |                |                |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| d <sub>7</sub> | d <sub>6</sub> | d <sub>5</sub> | d <sub>4</sub> | d <sub>3</sub> | d <sub>2</sub> | d <sub>1</sub> | d <sub>0</sub> |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|

 53H

**Description:** This is a 2-cycle instruction. Data in the accumulator is logically ANDed with an immediately-specified mask.

**Operation:** (A) ← (A) AND data

**Examples:** ANDID: ANL A,#0AFH

;AND' ACC CONTENTS

;WITH MASK 10101111

ANL A,#3 + X/Y

;AND' ACC CONTENTS

;WITH VALUE OF EXP

;3 + XY/Y'

**ANL BUS,#data\* Logical AND BUS with Immediate Mask**

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

|                |                |                |                |                |                |                |                |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| d <sub>7</sub> | d <sub>6</sub> | d <sub>5</sub> | d <sub>4</sub> | d <sub>3</sub> | d <sub>2</sub> | d <sub>1</sub> | d <sub>0</sub> |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|

 98H

**Description:** This is a 2-cycle instruction. Data on the BUS port is logically ANDed with an immediately-specified mask. This instruction assumes prior specification of an 'OUTL BUS, A' instruction.

**Operation:** (BUS) ← (BUS) AND data

**Example:** ANDBUS: ANL BUS,#MASK

;AND' BUS CONTENTS

;WITH MASK EQUAL VALUE

;OF SYMBOL 'MASK'

**ANL Pp,#data Logical AND Port 1-2 with Immediate Mask**

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 | 0 | p | p |
|---|---|---|---|---|---|---|---|

|                |                |                |                |                |                |                |                |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| d <sub>7</sub> | d <sub>6</sub> | d <sub>5</sub> | d <sub>4</sub> | d <sub>3</sub> | d <sub>2</sub> | d <sub>1</sub> | d <sub>0</sub> |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|

 99H-9AH

**Description:** This is a 2-cycle instruction. Data on port 'p' is logically ANDed with an immediately-specified mask.

**Operation:** (Pp) ← (Pp) AND DATA

p = 1-2

**Example:** ANDP2: ANL P2,#0F0H

;AND' PORT 2 CONTENTS

;WITH MASK 'F0' HEX

;(CLEAR P20-23)

\* For use with internal program memory ONLY.

**ANLD Pp,A Logical AND Port 4-7 with Accumulator Mask****Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 | 1 | p | p |
|---|---|---|---|---|---|---|---|

 9CH-9FH**Description:** This is a 2-cycle instruction. Data on port 'p' is logically ANDed with the digit mask contained in accumulator bits 0-3.**Operation:**  $(Pp) \leftarrow (Pp) \text{ AND } (A0-3)$  p = 4-7

Note: The mapping of port 'p' to opcode bits 0-1 is as follows:

| 10 | Port |
|----|------|
| 00 | 4    |
| 01 | 5    |
| 10 | 6    |
| 11 | 7    |

**Example:** ANDP4: ANLD P4,A ;'AND' PORT 4 CONTENTS  
;WITH ACC BITS 0-3**CALL address Subroutine Call****Encoding:**

|                 |                |                |   |   |   |   |   |
|-----------------|----------------|----------------|---|---|---|---|---|
| a <sub>10</sub> | a <sub>9</sub> | a <sub>8</sub> | 1 | 0 | 1 | 0 | 0 |
|-----------------|----------------|----------------|---|---|---|---|---|

|                |                |                |                |                |                |                |                |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| a <sub>7</sub> | a <sub>6</sub> | a <sub>5</sub> | a <sub>4</sub> | a <sub>3</sub> | a <sub>2</sub> | a <sub>1</sub> | a <sub>0</sub> |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|

Page Hex Op Code

|   |    |
|---|----|
| 0 | 14 |
| 1 | 34 |
| 2 | 54 |
| 3 | 74 |
| 4 | 94 |
| 5 | B4 |
| 6 | D4 |
| 7 | F4 |

**Description:** This is a 2-cycle instruction. The program counter and PSW bits 4-7 are saved in the stack. The stack pointer (PSW bits 0-2) is updated. Program control is then passed to the location specified by 'address'. PC bit 11 is determined by the most recent SEL MB instruction.

A CALL cannot begin in locations 2046-2047 or 4094-4095. Execution continues at the instruction following the CALL upon return from the subroutine.

**Operation:**  $((SP)) \leftarrow (PC), (PSW_{4-7})$   
 $(SP) \leftarrow (SP) + 1$   
 $(PC_{8-10}) \leftarrow (addr_{8-10})$   
 $(PC_{0-7}) \leftarrow (addr_{0-7})$   
 $(PC_{11}) \leftarrow DBF$



**Example:** Add three groups of two numbers. Put subtotals in locations 50, 51 and total in location 52.

```

MOV R0,#50      ;MOVE '50' DEC TO ADDRESS
                 ;REG 0
BEGADD: MOV A,R1  ;MOVE CONTENTS OF REG 1
                 ;TO ACC
ADD A,R2         ;ADD REG 2 TO ACC
CALL SUBTOT      ;CALL SUBROUTINE 'SUBTOT'
ADDC A,R3        ;ADD REG 3 TO ACC
ADDC A,R4        ;ADD REG 4 TO ACC
CALL SUBTOT      ;CALL SUBROUTINE 'SUBTOT'
ADDC A,R5        ;ADD REG 5 TO ACC
ADDC A,R6        ;ADD REG 6 TO ACC
CALL SUBTOT      ;CALL SUBROUTINE 'SUBTOT'
SUBTOT: MOV @R0,A ;MOVE CONTENTS OF ACC TO
                 ;LOCATION ADDRESSED BY
                 ;REG 0
INC R0           ;INCREMENT REG 0
RET              ;RETURN TO MAIN PROGRAM

```

#### CLR A Clear Accumulator

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 1 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 1 |
|---|---|---|---|

 27H

**Description:** The contents of the accumulator are cleared to zero.

**Operation:**  $A \leftarrow 0$

#### CLR C Clear Carry Bit

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 0 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 1 |
|---|---|---|---|

 97H

**Description:** During normal program execution, the carry bit can be set to one by the ADD, ADDC, RLC, CPL C, RRC, and DAA instructions. This instruction resets the carry bit to zero.

**Operation:**  $C \leftarrow 0$

#### CLR F1 Clear Flag 1

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 1 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
|---|---|---|---|

 A5H

**Description:** Flag 1 is cleared to zero.

**Operation:**  $(F1) \leftarrow 0$

## CLR F0 Clear Flag 0

Encoding: 

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
|---|---|---|---|

 85H

Description: Flag 0 is cleared to zero.

Operation:  $(F0) \leftarrow 0$

## CPL A Complement Accumulator

Encoding: 

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 1 |
|---|---|---|---|

 37H

Description: The contents of the accumulator are complemented. This is strictly a one's complement. Each one is changed to zero and vice-versa.

Operation:  $(A) \leftarrow \text{NOT}(A)$

Example: Assume accumulator contains 01101010.

CPLA: CPL A

;ACC CONTENTS ARE COMPLEMENTED TO 10010101

## CPL C Complement Carry Bit

Encoding: 

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 1 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 1 |
|---|---|---|---|

 A7H

Description: The setting of the carry bit is complemented; one is changed to zero, and zero is changed to one.

Operation:  $(C) \leftarrow \text{NOT}(C)$

Example: Set C to one; current setting is unknown.

CTO1: CLR C

;C IS CLEARED TO ZERO

CPL C

;C IS SET TO ONE

## CPL F0 Complement Flag 0

Encoding: 

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 0 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
|---|---|---|---|

 95H

Description: The setting of flag 0 is complemented; one is changed to zero, and zero is changed to one.

Operation:  $F0 \leftarrow \text{NOT}(F0)$

## CPL F1 Complement Flag 1

Encoding: 

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 1 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
|---|---|---|---|

 B5H

Description: The setting of flag 1 is complemented; one is changed to zero, and zero is changed to one.

Operation:  $(F1) \leftarrow \text{NOT}(F1)$

**DA A Decimal Adjust Accumulator**

**Encoding:** 0 1 0 1 0 1 1 1 57H

**Description:** The 8-bit accumulator value is adjusted to form two 4-bit Binary Coded Decimal (BCD) digits following the binary addition of BCD numbers. The carry bit C is affected. If the contents of bits 0-3 are greater than nine, or if AC is one, the accumulator is incremented by six.

The four high-order bits are then checked. If bits 4-7 exceed nine, or if C is one, these bits are increased by six. If an overflow occurs, C is set to one.

**Example:** Assume accumulator contains 10011011.  
DA A ;ACC Adjusted to 00000001  
;WITH C SET

|   |    |   |   |   |   |   |   |   |
|---|----|---|---|---|---|---|---|---|
| C | AC | 7 |   | 4 | 3 |   | 0 |   |
| 0 | 0  | 1 | 0 | 0 | 1 | 1 | 0 | 1 |
|   |    |   |   | 0 | 0 | 0 | 0 | 1 |
| 0 | 1  | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
|   |    |   |   | 0 | 1 | 1 | 0 |   |
| 1 | 0  | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

ADD SIX TO BITS 0-7  
ADD SIX TO BITS 4-7  
OVERFLOW TO C

**DEC A Decrement Accumulator**

**Encoding:** 0 0 0 0 0 1 1 1 07H

**Description:** The contents of the accumulator are decremented by one. The carry flag is not affected.

**Operation:** (A) ← (A) - 1

**Example:** Decrement contents of external data memory location 63.

MOV R0,#3FH ;MOVE '3F' HEX TO REG 0  
MOVX A, @R0 ;MOVE CONTENTS OF  
;LOCATION 63 TO ACC

DEC A ;DECREMENT ACC  
MOVX @R0,A ;MOVE CONTENTS OF ACC TO  
;LOCATION 63 IN EXPANDED  
;MEMORY

**DEC Rr Decrement Register**

**Encoding:** 1 1 0 0 1 r r r C8H-CFH

**Description:** The contents of working register 'r' are decremented by one.

**Operation:** (Rr) ← (Rr) - 1 r = 0-7

**Example:** DEC R1 ;DECREMENT CONTENTS OF REG 1

# DIS I External Interrupt

Encoding: 0 0 0 1 0 1 0 1 15H

Description: External interrupts are disabled. A low signal on the interrupt input pin has no effect.

# DIS TCNTI Disable Timer/Counter Interrupt

Encoding: 0 0 1 1 0 1 0 1 35H

Description: Timer/counter interrupts are disabled. Any pending timer interrupt request is cleared. The interrupt sequence is not initiated by an overflow, but the timer flag is set and time accumulation continues.

# DJNZ R<sub>r</sub>, address Decrement Register and Test

Encoding: 1 1 1 0 1 r r r a<sub>7</sub> a<sub>6</sub> a<sub>5</sub> a<sub>4</sub> a<sub>3</sub> a<sub>2</sub> a<sub>1</sub> a<sub>0</sub> E8H-EFH

Description: This is a 2-cycle instruction. Register 'r' is decremented, then tested for zero. If the register contains all zeros, program control falls through to the next instruction. If the register contents are not zero, control jumps to the specified 'address'.

The address in this case must evaluate to 8-bits, that is, the jump must be to a location within the current 256-location page.

Example: (Rr) ← (Rr) - 1 r = 0-7  
If Rr not 0  
(PC<sub>0-7</sub>) ← addr

Note: A 12-bit address specification does not cause an error if the DJNZ instruction and the jump target are on the same page. If the DJNZ instruction begins in location 255 of a page, it must jump to a target address on the following page.

Example: Increment values in data memory locations 50-54.

MOV R0,#50 ;MOVE '50' DEC TO ADDRESS

;REG 0

MOV R3,#5 ;MOVE '5' DEC TO COUNTER

;REG 3

INCRT: INC @R0

;INCREMENT CONTENTS OF

;LOCATION ADDRESSED BY

;REG 0

INC R0

;INCREMENT ADDRESS IN REG 0

DJNZ R3, INCRT

;DECREMENT REG 3 — JUMP TO

;'INCRT' IF REG 3 NONZERO

NEXT —

;'NEXT' ROUTINE EXECUTED

;IF R3 IS ZERO



**EN I Enable External Interrupt****Encoding:** 0 0 0 0 0 1 0 1 05H**Description:** External interrupts are enabled. A low signal on the interrupt input pin initiates the interrupt sequence.**EN TCNTI Enable Timer/Counter Interrupt****Encoding:** 0 0 1 0 0 1 0 1 25H**Description:** Timer/counter interrupts are enabled. An overflow of the timer/counter initiates the interrupt sequence.**ENT0 CLK Enable Clock Output****Encoding:** 0 1 1 1 0 1 0 1 75H**Description:** The test 0 pin is enabled to act as the clock output. This function is disabled by a system reset.**Example:** EMTST0: ENT0 CLK ;ENABLE T0 AS CLOCK OUTPUT**IN A,Pp Input Port or Data to Accumulator****Encoding:** 0 0 0 0 1 0 p p 09H-0AH**Description:** This is a 2-cycle instruction. Data present on port 'p' is transferred (read) to the accumulator.

**Operation:**  $(A) \leftarrow (Pp)$   $p = 1-2$

|                |                               |
|----------------|-------------------------------|
| INP12: IN A,P1 | ;INPUT PORT 1 CONTENTS TO ACC |
| MOV R6,A       | ;MOVE ACC CONTENTS TO REG 6   |
| IN A,P2        | ;INPUT PORT 2 CONTENTS TO ACC |
| MOV R7,A       | ;MOVE ACC CONTENTS TO REG 7   |

**INC A Increment Accumulator****Encoding:** 0 0 0 1 0 1 1 1 17H**Description:** The contents of the accumulator are incremented by one. Carry is not affected.**Operation:**  $(A) \leftarrow (A) + 1$

**Example:** Increment contents of location 100 in external data memory.

```
INCA: MOV R0,#100      ;MOVE '100' DEC TO ADDRESS REG 0
      MOVX A,@R0       ;MOVE CONTENTS OF LOCATION
                        ;100 TO ACC
      INC A            ;INCREMENT A
      MOVX @R0,A       ;MOVE ACC CONTENTS TO
                        ;LOCATION 101
```

### INC R<sub>r</sub> Increment Register

**Encoding:** 0 0 0 1 1 r r r 18H-1FH

**Description:** The contents of working register 'r' are incremented by one.

**Operation:** (Rr) ← (Rr) + 1 r = 0-7

**Example:** INCR0: INC R0 ;INCREMENT CONTENTS OF REG 0

### INC @R<sub>i</sub> Increment Data Memory Location

**Encoding:** 0 0 0 1 0 0 0 i 10H-11H

**Description:** The contents of the resident data memory location addressed by register 'i' bits 0-5\*\* are incremented by one.

**Operation:** ((Ri)) ← ((Ri)) + 1 i = 0-1

**Example:** INCDM: MOV R1,#03FH ;MOVE ONES TO REG 1  
INC @R1 ;INCREMENT LOCATION 63

### INS A,BUS\* Strobed Input of BUS Data to Accumulator

**Encoding:** 0 0 0 0 1 0 0 0 08H

**Description:** This is a 2-cycle instruction. Data present on the BUS port is transferred (read) to the accumulator when the RD pulse is dropped. (Refer to section on programming memory expansion for details.)

**Operation:** (A) ← (BUS)

**Example:** INPBUS: INS A,BUS ;INPUT BUS CONTENTS TO ACC

\* For use with internal program memory ONLY.

\*\* 0-5 in 8048AH/8748H

0-6 in 8049AH/8749H

0-7 in 8050AH

**JBb address Jump If Accumulator Bit Is Set**

Encoding: 

|                |                |                |   |   |   |   |   |
|----------------|----------------|----------------|---|---|---|---|---|
| b <sub>2</sub> | b <sub>1</sub> | b <sub>0</sub> | 1 | 0 | 0 | 1 | 0 |
|----------------|----------------|----------------|---|---|---|---|---|

|                |                |                |                |                |                |                |                |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| a <sub>7</sub> | a <sub>6</sub> | a <sub>5</sub> | a <sub>4</sub> | a <sub>3</sub> | a <sub>2</sub> | a <sub>1</sub> | a <sub>0</sub> |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|

| Accumulator Bit | Hex Op Code |
|-----------------|-------------|
| 0               | 12          |
| 1               | 32          |
| 2               | 52          |
| 3               | 72          |
| 4               | 92          |
| 5               | B2          |
| 6               | D2          |
| 7               | F2          |

**Description:** This is a 2-cycle instruction. Control passes to the specified address if accumulator bit 'b' is set to one.

**Operation:** b = 0-7

(PC<sub>0-7</sub>) ← addr

(PC) = (PC) + 2

**Example:** JB4IS1: JB4 NEXT

If Bb = 1

If Bb = 0

;JUMP TO 'NEXT' ROUTINE

;IF ACC BIT 4 = 1

**JC address Jump If Carry Is Set**

Encoding: 

|   |   |   |   |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 0 |
|---|---|---|---|

|                |                |                |                |                |                |                |                |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| a <sub>7</sub> | a <sub>6</sub> | a <sub>5</sub> | a <sub>4</sub> | a <sub>3</sub> | a <sub>2</sub> | a <sub>1</sub> | a <sub>0</sub> |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|

 F6H

**Description:** This is a 2-cycle instruction. Control passes to the specified address if the carry bit is set to one.

**Operation:** (PC<sub>0-7</sub>) ← addr

(PC) = (PC) + 2

**Example:** JC1: JC OVFLOW

If C = 1

If C = 0

;JUMP TO 'OVFLOW' ROUTINE

;IF C = 1

**JF0 address Jump If Flag 0 Is Set**

Encoding: 

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 1 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 0 |
|---|---|---|---|

|                |                |                |                |                |                |                |                |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| a <sub>7</sub> | a <sub>6</sub> | a <sub>5</sub> | a <sub>4</sub> | a <sub>3</sub> | a <sub>2</sub> | a <sub>1</sub> | a <sub>0</sub> |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|

 B6H

**Description:** This is a 2-cycle instruction. Control passes to the specified address if flag 0 is set to one.

**Operation:** (PC<sub>0-7</sub>) ← addr

(PC) = (PC) + 2

**Example:** JF0IS1: JF0 TOTAL

If F0 = 1

If F0 = 0

;JUMP TO 'TOTAL' ROUTINE IF F0 = 1

**JF1 address Jump If Flag 1 Is Set**

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 0 |
|---|---|---|---|

|                |                |                |                |
|----------------|----------------|----------------|----------------|
| a <sub>7</sub> | a <sub>6</sub> | a <sub>5</sub> | a <sub>4</sub> |
|----------------|----------------|----------------|----------------|

|                |                |                |                |
|----------------|----------------|----------------|----------------|
| a <sub>3</sub> | a <sub>2</sub> | a <sub>1</sub> | a <sub>0</sub> |
|----------------|----------------|----------------|----------------|

 76H

**Description:** This is a 2-cycle instruction. Control passes to the specified address if flag 1 is set to one.

**Operation:** (PC<sub>0-7</sub>) ← addr If F1 = 1  
(PC) = (PC + 2) If F1 = 0

**Example:** JF1IS1: JF1 FILBUF ;JUMP TO 'FILBUF'  
;ROUTINE IF F1 = 1

**JMP address Direct Jump within 2K Block**

**Encoding:**

|                 |                |                |   |
|-----------------|----------------|----------------|---|
| a <sub>10</sub> | a <sub>9</sub> | a <sub>8</sub> | 0 |
|-----------------|----------------|----------------|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
|---|---|---|---|

|                |                |                |                |
|----------------|----------------|----------------|----------------|
| a <sub>7</sub> | a <sub>6</sub> | a <sub>5</sub> | a <sub>4</sub> |
|----------------|----------------|----------------|----------------|

|                |                |                |                |
|----------------|----------------|----------------|----------------|
| a <sub>3</sub> | a <sub>2</sub> | a <sub>1</sub> | a <sub>0</sub> |
|----------------|----------------|----------------|----------------|

Page Hex Op Code

|   |    |
|---|----|
| 0 | 04 |
| 1 | 24 |
| 2 | 44 |
| 3 | 64 |
| 4 | 84 |
| 5 | A4 |
| 6 | C4 |
| 7 | E4 |

**Description:** This is a 2-cycle instruction. Bits 0-10 of the program counter are replaced with the directly-specified address. The setting of PC bit 11 is determined by the most recent SELECT MB instruction.

**Operation:** (PC<sub>8-10</sub>) ← addr 8-10  
(PC<sub>0-7</sub>) ← addr 0-7  
(PC<sub>11</sub>) ← DBF

**Example:** JMP SUBTOT ;JUMP TO SUBROUTINE 'SUBTOT'  
JMP \$-6 ;JUMP TO INSTRUCTION SIX  
;LOCATIONS BEFORE CURRENT  
;LOCATION  
JMP 2FH ;JUMP TO ADDRESS '2F' HEX

**JMPP @A Indirect Jump within Page**

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 1 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
|---|---|---|---|

 B3H

**Description:** This is a 2-cycle instruction. The contents of the program memory location pointed to by the accumulator are substituted for the 'page' portion of the program counter (PC bits 0-7).



**Operation:**  $(PC_{0-7}) \leftarrow ((A))$

**Example:** Assume accumulator contains 0FH.

JMPPAG: JMPP @A ;JUMP TO ADDRESS STORED IN  
;LOCATION 15 IN CURRENT PAGE

#### JNC address Jump If Carry Is Not Set

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 1 | 1 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 0 |
|---|---|---|---|

|                |                |                |                |                |                |                |                |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| a <sub>7</sub> | a <sub>6</sub> | a <sub>5</sub> | a <sub>4</sub> | a <sub>3</sub> | a <sub>2</sub> | a <sub>1</sub> | a <sub>0</sub> |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|

 E6H

**Description:** This is a 2-cycle instruction. Control passes to the specified address if the carry bit is not set, that is, equals zero.

**Operation:**  $(PC_{0-7}) \leftarrow \text{addr}$  If C = 0  
 $(PC) = (PC) + 2$  If C = 1

**Example:** JC0: JNC NOVFO ;JUMP TO 'NOVFO' ROUTINE  
;IF C = 0

#### JNI address Jump If Interrupt Input Is Low

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 0 |
|---|---|---|---|

|                |                |                |                |                |                |                |                |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| a <sub>7</sub> | a <sub>6</sub> | a <sub>5</sub> | a <sub>4</sub> | a <sub>3</sub> | a <sub>2</sub> | a <sub>1</sub> | a <sub>0</sub> |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|

 86H

**Description:** This is a 2-cycle instruction. Control passes to the specified address if the interrupt input signal is low (= 0), that is, an external interrupt has been signaled. (This signal initiates an interrupt service sequence if the external interrupt is enabled.)

**Operation:**  $(PC_{0-7}) \leftarrow \text{addr}$  If I = 0  
 $(PC) = (PC) + 2$  If I = 1

**Example:** LOC 3: JN1 EXTINT ;JUMP TO 'EXTINT' ROUTINE  
;IF I = 0

#### JNT0 address Jump If Test 0 Is Low

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 1 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 0 |
|---|---|---|---|

|                |                |                |                |                |                |                |                |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| a <sub>7</sub> | a <sub>6</sub> | a <sub>5</sub> | a <sub>4</sub> | a <sub>3</sub> | a <sub>2</sub> | a <sub>1</sub> | a <sub>0</sub> |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|

 26H

**Description:** This is a 2-cycle instruction. Control passes to the specified address, if the test 0 signal is low.

**Operation:**  $(PC_{0-7}) \leftarrow \text{addr}$  If T0 = 0  
 $(PC) = (PC) + 2$  If T0 = 1

**Example:** JT0LOW: JNT0 60 ;JUMP TO LOCATION 60 DEC  
;IF T0 = 0

# MCS®-48 INSTRUCTION SET

## JNT1 address Jump If Test 1 Is Low

**Encoding:** 0 1 0 0 0 1 1 0 a<sub>7</sub> a<sub>6</sub> a<sub>5</sub> a<sub>4</sub> a<sub>3</sub> a<sub>2</sub> a<sub>1</sub> a<sub>0</sub> 46H

**Description:** This is a 2-cycle instruction. Control passes to the specified address, if the test 1 signal is low.

**Operation:** (PC<sub>0-7</sub>) ← addr If T1 = 0  
(PC) = (PC) + 2 If T1 = 1

## JNZ Address Jump If Accumulator Is Not Zero

**Encoding:** 1 0 0 1 0 1 1 0 a<sub>7</sub> a<sub>6</sub> a<sub>5</sub> a<sub>4</sub> a<sub>3</sub> a<sub>2</sub> a<sub>1</sub> a<sub>0</sub> 96H

**Description:** This is a 2-cycle instruction. Control passes to the specified address if the accumulator contents are nonzero at the time this instruction is executed.

**Operation:** (PC<sub>0-7</sub>) ← addr If A ≠ 0  
(PC) = (PC) + 2 If A = 0

**Example:** JACCN0: JNZ 0ABH ;JUMP TO LOCATION 'AB' HEX  
;IF ACC VALUE IS NONZERO

## JTF address Jump If Timer Flag Is Set

**Encoding:** 0 0 0 1 0 1 1 0 a<sub>7</sub> a<sub>6</sub> a<sub>5</sub> a<sub>4</sub> a<sub>3</sub> a<sub>2</sub> a<sub>1</sub> a<sub>0</sub> 16H

**Description:** This is a 2-cycle instruction. Control passes to the specified address if the timer flag is set to one, that is, the timer/counter register has overflowed. Testing the timer flag resets it to zero. (This overflow initiates an interrupt service sequence if the timer-overflow interrupt is enabled.)

**Operation:** (PC<sub>0-7</sub>) ← addr If TF = 1  
(PC) = (PC) + 2 If TF = 0

**Example:** JTF1: JTF TIMER ;JUMP TO 'TIMER' ROUTINE  
;IF TF = 1

## JT0 address Jump If Test 0 Is High

**Encoding:** 0 0 1 1 0 1 1 0 a<sub>7</sub> a<sub>6</sub> a<sub>5</sub> a<sub>4</sub> a<sub>3</sub> a<sub>2</sub> a<sub>1</sub> a<sub>0</sub> 36H

**Description:** This is a 2-cycle instruction. Control passes to the specified address if the test 0 signal is high (= 1).

**Operation:** (PC<sub>0-7</sub>) ← addr If T0 = 1  
(PC) = (PC) + 2 If T0 = 0

**Example:** JT0HI: JT0 53 ;JUMP TO LOCATION 53 DEC  
;IF T0 = 1

# MCS®-48 INSTRUCTION SET

## JT1 address Jump If Test 1 Is High

**Encoding:** 0 1 0 1 0 1 1 0 a<sub>7</sub> a<sub>6</sub> a<sub>5</sub> a<sub>4</sub> a<sub>3</sub> a<sub>2</sub> a<sub>1</sub> a<sub>0</sub> 56H

**Description:** This is a 2-cycle instruction. Control passes to the specified address if the test 1 signal is high (= 1).

**Operation:** (PC<sub>0-7</sub>) ← addr If T1 = 1  
(PC) = (PC) + 2 If T1 = 0

**Example:** JT1HI: JT1 COUNT ;JUMP TO 'COUNT' ROUTINE  
;IF T1 = 1

## JZ address Jump If Accumulator Is Zero

**Encoding:** 1 1 0 0 0 1 1 0 a<sub>7</sub> a<sub>6</sub> a<sub>5</sub> a<sub>4</sub> a<sub>3</sub> a<sub>2</sub> a<sub>1</sub> a<sub>0</sub> C6H

**Description:** This is a 2-cycle instruction. Control passes to the specified address if the accumulator contains all zeros at the time this instruction is executed.

**Operation:** (PC<sub>0-7</sub>) ← addr If A = 0  
(PC) = (PC) + 2 If A ≠ 1

**Example:** JACCO: JZ 0A3H ;JUMP TO LOCATION 'A3' HEX  
;IF ACC VALUE IS ZERO

## MOV A,#data Move Immediate Data to Accumulator

**Encoding:** 0 0 1 0 0 0 1 1 a<sub>7</sub> a<sub>6</sub> a<sub>5</sub> a<sub>4</sub> a<sub>3</sub> a<sub>2</sub> a<sub>1</sub> a<sub>0</sub> 23H

**Description:** This is a 2-cycle instruction. The 8-bit value specified by 'data' is loaded in the accumulator.

**Operation:** (A) ← data

**Example:** MOV A,#0A3H ;MOVE 'A3' HEX TO ACC

## MOV A,PSW Move PSW Contents to Accumulator

**Encoding:** 1 1 0 0 0 1 1 1 C7H

**Description:** The contents of the program status word are moved to the accumulator.

**Operation:** (A) ← (PSW)

**Example:** Jump to 'RB1SET' routine if PSW bank switch, bit 4, is set.

BSCHK: MOV A,PSW ;MOVE PSW CONTENTS TO ACC

JB4 RB1SET ;JUMP TO 'RB1SET' IF ACC BIT 4 = 1

**MOV A,R<sub>r</sub> Move Register Contents to Accumulator**

Encoding: 

|   |   |   |   |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 1 | r | r | r |
|---|---|---|---|

 F8H-FFH

Description: 8-bits of data are removed from working register 'r' into the accumulator.

Operation:  $(A) \leftarrow (R_r)$   $r = 0-7$

Example: MAR: MOV A,R3 ;MOVE CONTENTS OF REG 3 TO ACC

**MOV A,@R<sub>i</sub> Move Data Memory Contents to Accumulator**

Encoding: 

|   |   |   |   |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 0 | i |
|---|---|---|---|

 F0H-F1H

Description: The contents of the resident data memory location addressed by bits 0-5\*\* of register 'i' are moved to the accumulator. Register 'i' contents are unaffected.

Operation:  $(A) \leftarrow ((R_i))$   $i = 0-1$

Example: Assume R1 contains 00110110.

MADM: MOV A,@R1 ;MOVE CONTENTS OF DATA MEM  
;LOCATION 54 TO ACC

**MOV A,T Move Timer/Counter Contents to Accumulator**

Encoding: 

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 1 | 0 |
|---|---|---|---|

 42H

Description: The contents of the timer/event-counter register are moved to the accumulator.

Operation:  $(A) \leftarrow (T)$

Example: Jump to "EXIT" routine when timer reaches '64', that is, when bit 6 set—  
assuming initialization 64,

TIMCHK: MOV A,T ;MOVE TIMER CONTENTS TO ACC  
JB6 EXIT ;JUMP TO 'EXIT' IF ACC BIT 6 = 1

**MOV PSW,A Move Accumulator Contents to PSW**

Encoding: 

|   |   |   |   |
|---|---|---|---|
| 1 | 1 | 0 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 1 |
|---|---|---|---|

 D7H

Description: The contents of the accumulator are moved into the program status word. All condition bits and the stack pointer are affected by this move.

Operation:  $(PSW) \leftarrow (A)$

Example: Move up stack pointer by two memory locations, that is, increment the  
pointer by one.

INCPTR: MOV A,PSW ;MOVE PSW CONTENTS TO ACC  
INC A ;INCREMENT ACC BY ONE  
MOV PSW,A ;MOVE ACC CONTENTS TO PSW

\*\* 0-5 in 8048AH/8748H

0-6 in 8049AH/8749H

0-7 in 8050AH



**MOV R<sub>r</sub>,A Move Accumulator Contents to Register**

Encoding: 

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 1 | r | r | r |
|---|---|---|---|---|---|---|---|

 A8H-AFH

Description: The contents of the accumulator are moved to register 'r'.

Operation:  $(Rr) \leftarrow (A)$   $r = 0-7$

Example: MRA: MOV R0,A ;MOVE CONTENTS OF ACC TO REG 0

**MOV R<sub>r</sub>,#data Move Immediate Data to Register**

Encoding: 

|   |   |   |   |   |                |                |                |
|---|---|---|---|---|----------------|----------------|----------------|
| 1 | 0 | 1 | 1 | 1 | r <sub>2</sub> | r <sub>1</sub> | r <sub>0</sub> |
|---|---|---|---|---|----------------|----------------|----------------|

|                |                |                |                |                |                |                |                |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| d <sub>7</sub> | d <sub>6</sub> | d <sub>5</sub> | d <sub>4</sub> | d <sub>3</sub> | d <sub>2</sub> | d <sub>1</sub> | d <sub>0</sub> |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|

 B8H-BFH

Description: This is a 2-cycle instruction. The 8-bit value specified by 'data' is moved to register 'r'.

Operation:  $(Rr) \leftarrow \text{data}$   $r = 0-7$

Examples: MIR4: MOV R4,#HEXTEN ;THE VALUE OF THE SYMBOL  
;HEXTEN' IS MOVED INTO REG 4  
MIR 5: MOV R5,#PI\*(R\*R) ;THE VALUE OF THE EXPRESSION  
;'PI\*(R\*R)' IS MOVED INTO REG 5  
MIR 6: MOV R6,#0ADH ;'AD' HEX IS MOVED INTO REG 6

**MOV @ R<sub>i</sub>,A Move Accumulator Contents to Data Memory**

Encoding: 

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | i |
|---|---|---|---|---|---|---|---|

 A0H-A1H

Description: The contents of the accumulator are moved to the resident data memory location whose address is specified by bits 0-5\*\* of register 'i'. Register 'i' contents are unaffected.

Operation:  $((Ri)) \leftarrow (A)$   $i = 0-1$

Example: Assume R0 contains 00000111.  
MDMA: MOV @R0,A ;MOVE CONTENTS OF ACC TO  
;LOCATION 7 (REG 7)

**MOV @ R<sub>i</sub>,#data Move Immediate Data to Data memory**

Encoding: 

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | i |
|---|---|---|---|---|---|---|---|

|                |                |                |                |                |                |                |                |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| d <sub>7</sub> | d <sub>6</sub> | d <sub>5</sub> | d <sub>4</sub> | d <sub>3</sub> | d <sub>2</sub> | d <sub>1</sub> | d <sub>0</sub> |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|

 B0H-B1H

Description: This is a 2-cycle instruction. The 8-bit value specified by 'data' is moved to the resident data memory location addressed by register 'i', bits 0-5\*\*.

Operation:  $((Ri)) \leftarrow \text{data}$   $i = 0-1$

Examples: Move the hexadecimal value AC3F to locations 62-63.  
MIDM: MOV R0,#62 ;MOVE '62' DEC TO ADDR REG 0  
MOV @R0,#0ACH ;MOVE 'AC' HEX TO LOCATION 62  
INC R0 ;INCREMENT REG 0 to '63'  
MOV @R0,#3FH ;MOVE '3F' HEX TO LOCATION 63

\*\* 0-5 in 8048AH/8748H  
0-6 in 8049AH/8749H  
0-7 in 8050AH

**MOV T,A Move Accumulator Contents to Timer/Counter**

Encoding: 

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 1 | 0 |
|---|---|---|---|

 62H

**Description:** The contents of the accumulator are moved to the timer/event-counter register.

**Operation:** (T) ← (A)

**Example:** Initialize and start event counter.

```

INITEC: CLR A           ;CLEAR ACC TO ZEROS
        MOV T,A         ;MOVE ZEROS TO EVENT COUNTER
        START CNT       ;START COUNTER

```

**MOVD A,Pp Move Port 4-7 Data to Accumulator**

Encoding: 

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 1 | 1 | p | p |
|---|---|---|---|

 0CH-0FH

**Description:** This is a 2-cycle instruction. Data on 8243 port 'p' is moved (read) to accumulator bits 0-3. Accumulator bits 4-7 are zeroed.

**Operation:** (0-3) ← (Pp)                      p = 4-7  
(4-7) ← 0

**Note:** Bits 0-7 of the opcode are used to represent ports 4-7. If you are coding in binary rather than assembly language, the mapping is as follows:

| Bits 1 0 | Port |
|----------|------|
| 0 0      | 4    |
| 0 1      | 5    |
| 1 0      | 6    |
| 1 1      | 7    |

**Example:** INPPT5: MOVD A,P5                      ;MOVE PORT 5 DATA TO ACC  
   ;BITS 0-3, ZERO ACC BITS 4-7

**MOVD Pp,A Move Accumulator Data to Port 4-7**

Encoding: 

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 1 | 1 | p | p |
|---|---|---|---|

 3CH-3FH

**Description:** This is a 2-cycle instruction. Data in accumulator bits 0-3 is moved (written) to 8243 port 'p'. Accumulator bits 4-7 are unaffected. (See NOTE above regarding port mapping.)

**Operation:** (Pp) ← (A<sub>0-3</sub>)                      P = 4-7

**Example:** Move data in accumulator to ports 4 and 5.

```

OUTP45: MOVD P4,A       ;MOVE ACC BITS 0-3 TO PORT 4
        SWAP A           ;EXCHANGE ACC BITS 0-3 and 4-7
        MOVD P5,A       ;MOVE ACC BITS 0-3 TO PORT 5

```

**MOVP A,@A Move Current Page Data to Accumulator****Encoding:** 1 0 1 0 0 0 1 1 A3H

**Description:** The contents of the program memory location addressed by the accumulator are moved to the accumulator. Only bits 0-7 of the program counter are affected, limiting the program memory reference to the current page. The program counter is restored *following* this operation.

**Operation:**  $(PC_{0-7}) \leftarrow (A)$   
 $(A) \leftarrow ((PC))$

Note: This is a 1-byte, 2-cycle instruction. If it appears in location 255 of a program memory page, @A addresses a location in the *following* page.

**Example:** MOV128: MOV A,#128 ;MOVE '128' DEC TO ACC  
 MOV P A,@A ;CONTENTS OF 129th LOCATION IN  
 ;CURRENT PAGE ARE MOVED TO ACC

**MOV P3 A,@A Move Page 3 Data to Accumulator****Encoding:** 1 1 1 0 0 0 1 1 E3H

**Description:** This is a 2-cycle instruction. The contents of the program memory location (within page 3) addressed by the accumulator are moved to the accumulator. The program counter is restored following this operation.

**Operation:**  $(PC_{0-7}) \leftarrow (A)$   
 $(PC_{8-11}) \leftarrow 0011$   
 $(A) \leftarrow ((PC))$

**Example:** Look up ASCII equivalent of hexadecimal code in table contained at the beginning of page 3. Note that ASCII characters are designated by a 7-bit code; the eighth bit is always reset.

TABSCH: MOV A,#0B8H ;MOVE 'B8' HEX TO ACC (10111000)  
 ANL A,#7FH ;LOGICAL AND ACC TO MASK BIT  
 ;7 (00111000)  
 MOV P3 A,@A ;MOVE CONTENTS OF LOCATION '38'  
 ;HEX IN PAGE 3 TO ACC (ASCII '8')

Access contents of location in page 3 labelled TAB1.

Assume current program location is not in page 3.

TABSCH: MOV A,#LOW TAB 1 ;ISOLATE BITS 0-7 OF LABEL

MOV P3 A,@A ;MOVE CONTENTS OF PAGE 3  
 ;LOCATION LABELED 'TAB1' TO ACC

### MOVX A,@R<sub>i</sub> Move External-Data-Memory Contents to Accumulator

Encoding: 1 0 0 0 0 0 0 i 80H-81H

**Description:** This is a 2-cycle instruction. The contents of the external data memory location addressed by register 'i' are moved to the accumulator. Register 'i' contents are unaffected. A read pulse is generated.

**Operation:** (A) ← ((R<sub>i</sub>)) i = 0-1

**Example:** Assume R1 contains 01110110.

MAXDM: MOVX A,@R1 ;MOVE CONTENTS OF LOCATION  
;118 TO ACC

### MOVX @R<sub>i</sub>,A Move Accumulator Contents to External Data Memory

Encoding: 1 0 0 1 0 0 0 i 90H-91H

**Description:** This is a 2-cycle instruction. The contents of the accumulator are moved to the external data memory location addressed by register 'i'. Register 'i' contents are unaffected. A write pulse is generated.

**Operation:** ((R<sub>i</sub>)) ← A i = 0-1

**Example:** Assume R0 contains 11000111.

MXDMA: MOVX @R0,A ;MOVE CONTENTS OF ACC TO  
;LOCATION 199 IN EXPANDED  
;DATA MEMORY

### NOP The NOP Instruction

Encoding: 0 0 0 0 0 0 0 0 00H

**Description:** No operation is performed. Execution continues with the following instruction.

### ORL A,R<sub>r</sub> Logical OR Accumulator With Register Mask

Encoding: 0 1 0 0 1 r r r 48H-4FH

**Description:** Data in the accumulator is logically ORed with the mask contained in working register 'r'.

**Operation:** (A) ← (A) OR (R<sub>r</sub>) r = 0-7

**Example:** ORREG: ORL A,R4

; 'OR' ACC CONTENTS WITH  
; MASK IN REG 4



**ORL A,@R<sub>i</sub> Logical OR Accumulator With Memory Mask**

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 0 | i |
|---|---|---|---|

 40H-41H

**Description:** Data in the accumulator is logically ORed with the mask contained in the resident data memory location referenced by register 'i', bits 0-5\*\*.

**Operation:**  $(A) \leftarrow (A) \text{ OR } ((R_i))$   $i = 0-1$

**Example:** ORDM: MOV R0,#3FH ;MOVE '3F' HEX TO REG 0  
                   ORL A,@R0 ;'OR' AC CONTENTS WITH MASK  
                               IN LOCATION 63

**ORL A,#data Logical OR Accumulator With Immediate Mask**

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
|---|---|---|---|

|                |                |                |                |
|----------------|----------------|----------------|----------------|
| d <sub>7</sub> | d <sub>6</sub> | d <sub>5</sub> | d <sub>4</sub> |
|----------------|----------------|----------------|----------------|

|                |                |                |                |
|----------------|----------------|----------------|----------------|
| d <sub>3</sub> | d <sub>2</sub> | d <sub>1</sub> | d <sub>0</sub> |
|----------------|----------------|----------------|----------------|

 43H

**Description:** This is a 2-cycle instruction. Data in the accumulator is logically ORed with an immediately-specified mask.

**Operation:**  $(A) \leftarrow (A) \text{ OR data}$

**Example:** ORID: ORL A,#'X' ;'OR' ACC CONTENTS WITH MASK  
                               01011000 (ASCII VALUE OF 'X')

**ORL BUS,#data\* Logical OR BUS With Immediate Mask**

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
|---|---|---|---|

|                |                |                |                |
|----------------|----------------|----------------|----------------|
| d <sub>7</sub> | d <sub>6</sub> | d <sub>5</sub> | d <sub>4</sub> |
|----------------|----------------|----------------|----------------|

|                |                |                |                |
|----------------|----------------|----------------|----------------|
| d <sub>3</sub> | d <sub>2</sub> | d <sub>1</sub> | d <sub>0</sub> |
|----------------|----------------|----------------|----------------|

 88H

**Description:** This is a 2-cycle instruction. Data on the BUS port is logically ORed with an immediately-specified mask. This instruction assumes prior specification on an 'OUTL BUS,A' instruction.

**Operation:**  $(BUS) \leftarrow (BUS) \text{ OR data}$

**Example:** ORBUS: ORL BUS,#HEXMSK ;'OR' BUS CONTENTS WITH MASK  
   ;EQUAL VALUE OF SYMBOL 'HEXMSK'

**ORL Pp,#data Logical OR Port 1 or 2 With Immediate Mask**

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | p | p |
|---|---|---|---|

|                |                |                |                |
|----------------|----------------|----------------|----------------|
| d <sub>7</sub> | d <sub>6</sub> | d <sub>5</sub> | d <sub>4</sub> |
|----------------|----------------|----------------|----------------|

|                |                |                |                |
|----------------|----------------|----------------|----------------|
| d <sub>3</sub> | d <sub>2</sub> | d <sub>1</sub> | d <sub>0</sub> |
|----------------|----------------|----------------|----------------|

 89H-8AH

**Description:** This is a 2-cycle instruction. Data on port 'p' is logically ORed with an immediately-specified mask.

**Operation:**  $(Pp) \leftarrow (Pp) \text{ OR data}$   $p = 1-2$

**Example:** ORP1: ORL P1,#0FFH ;'OR' PORT 1 CONTENTS WITH MASK  
                                   ;'FF' HEX (SET PORT 1 TO ALL ONES)

\* For use with internal program memory ONLY.

\*\* 0-5 in 8048AH/8748H

0-6 in 8049AH/8749H

0-7 in 8050AH

**ORLD Pp,A Logical OR Port 4-7 With Accumulator Mask****Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 1 | 1 | p | p |
|---|---|---|---|

 8CH-8FH**Description:** This is a 2-cycle instruction. Data on port 'p' is logically ORed with the digit mask contained in accumulator bits 0-3.**Operation:**  $(Pp) \leftarrow (Pp) \text{ OR } (A_{0-3})$  p = 4-7**Example:** ORP7: ORLD P7,A ;'OR' PORT 7 CONTENTS WITH ACC  
;BITS 0-3**OUTL BUS,A\* Output Accumulator Data to BUS****Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 1 | 0 |
|---|---|---|---|

 02H**Description:** This is a 2-cycle instruction. Data residing in the accumulator is transferred (written) to the BUS port and latched. The latched data remains valid until altered by another OUTL instruction. Any other instruction requiring use of the BUS port (except INS) destroys the contents of the BUS latch. This includes expanded memory operations (such as the MOVX instruction). Logical operations on BUS data (AND, OR) assume the OUTL BUS,A instruction has been issued previously.**Operation:**  $(\text{BUS}) \leftarrow (A)$ **Example:** OUTLBP: OUTL BUS, A ;OUTPUT ACC CONTENTS TO BUS**OUTL Pp,A Output Accumulator Data to Port 1 or 2****Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | p | p |
|---|---|---|---|

 39H-3AH**Description:** This is a 2-cycle instruction. Data residing in the accumulator is transferred (written) to port 'p' and latched.**Operation:**  $(Pp) \leftarrow (A)$  p = 1-2**Example:** OUTLP: MOV A,R7 ;MOVE REG 7 CONTENTS TO ACC  
OUTL P2,A ;OUTPUT ACC CONTENTS TO PORT 2  
MOV A, R6 ;MOV REG 6 CONTENTS TO ACC  
OUTL P1,A ;OUTPUT ACC CONTENTS TO PORT 1

\* For use with internal program memory ONLY.

**RET Return Without PSW Restore****Encoding:** 1 0 0 0 0 0 1 1 83H**Description:** This is a 2-cycle instruction. The stack pointer (PSW bits 0-2) is decremented. The program counter is then restored from the stack. PSW bits 4-7 are not restored.**Operation:** (SP) ← (SP)-1  
(PC) ← ((SP))**RETR Return with PSW Restore****Encoding:** 1 0 0 1 0 0 1 1 93H**Description:** This is a 2-cycle instruction. The stack pointer is decremented. The program counter and bits 4-7 of the PSW are then restored from the stack. Note that RETR should be used to return from an interrupt, but should not be used within the interrupt service routine as it signals the end of an interrupt routine by resetting the Interrupt in Progress flip-flop.**Operation:** (SP) ← (SP)-1  
(PC) ← ((SP))  
(PSW 4-7) ← ((SP))

**RL A Rotate Left without Carry****Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 1 | 1 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 1 |
|---|---|---|---|

 E7H**Description:** The contents of the accumulator are rotated left one bit. Bit 7 is rotated into the bit 0 position.**Operation:**  $(A_{n+1}) \leftarrow (A_n)$   
 $(A_0) \leftarrow (A_7)$   $n = 0-6$ **Example:** Assume accumulator contains 10110001.  
RLNC: RL A ;NEW ACC CONTENTS ARE 01100011**RLC A Rotate Left through Carry****Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 1 |
|---|---|---|---|

 F7H**Description:** The contents of the accumulator are rotated left one bit. Bit 7 replaces the carry bit; the carry bit is rotated into the bit 0 position.**Operation:**  $(A_{n+1}) \leftarrow (A_n)$   
 $n = 0-6$   
 $(A_0) \leftarrow (C)$   
 $(C) \leftarrow (A_7)$ **Example:** Assume accumulator contains a 'signed' number; isolate sign without changing value.

|  |  |
|--|--|
| RLTC: CLR C ;CLEAR CARRY TO ZERO<br>RLC A ;ROTATE ACC LEFT, SIGN<br>;BIT (7) IS PLACED IN CARRY<br>RR A ;ROTATE ACC RIGHT — VALUE<br>;(BITS 0-6) IS RESTORED,<br>;CARRY UNCHANGED, BIT 7<br>;IS ZERO |  |
|--|--|

**RR A Rotate Right without Carry****Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 1 |
|---|---|---|---|

 77H**Description:** The contents of the accumulator are rotated right one bit. Bit 0 is rotated into the bit 7 position.**Operation:**  $(A_n) \leftarrow (A_{n+1})$   $n = 0-6$   
 $(A_7) \leftarrow (A_0)$ **Example:** Assume accumulator contains 10110001.  
RRNC: RR A ;NEW ACC CONTENTS ARE 11011000



### RRC A Rotate Right through Carry

Encoding: 0 1 1 0 0 1 1 1 67H

**Description:** The contents of the accumulator are rotated right one bit. Bit 0 replaces the carry bit; the carry bit is rotated into the bit 7 position.

**Operation:**  $(A_n) \leftarrow (A_{n+1})$   $n = 0-6$   
 $(A_7) \leftarrow (C)$   
 $(C) \leftarrow (A_0)$

**Example:** Assume carry is not set and accumulator contains 10110001.  
RRTC: RRC A ;CARRY IS SET AND ACC  
;CONTAINS 01011000

### SEL MB0 Select Memory Bank 0

Encoding: 1 1 1 0 0 1 0 1 E5H

**Description:** PC bit 11 is set to zero on next JMP or CALL instruction. All references to program memory addresses fall within the range 0-2047.

**Operation:**  $(DBF) \leftarrow 0$

**Example:** Assume program counter contains 834 Hex.  
SEL MB0 ;SELECT MEMORY BANK 0  
JMP \$+20 ;JUMP TO LOCATION 58 HEX

### SEL MB1 Select Memory Bank 1

Encoding: 1 1 1 1 0 1 0 1 F5H

**Description:** PC bit 11 is set to one on next JMP or CALL instruction. All references to program memory addresses fall within the range 2048-4095.

**Operation:**  $(DBF) \leftarrow 1$

**SEL RB0 Select Register Bank 0**Encoding: 

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|

 C5H

**Description:** PSW bit 4 is set to zero. References to working registers 0-7 address data memory locations 0-7. This is the recommended setting for normal program execution.

**Operation:** (BS) ← 0

**SEL RB1 Select Register Bank 1**Encoding: 

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|

 D5H

**Description:** PSW bit 4 is set to one. References to working registers 0-7 address data memory locations 24-31. This is the recommended setting for interrupt service routines, since locations 0-7 are left intact. The setting of PSW bit 4 in effect at the time of an interrupt is restored by the RETR instruction when the interrupt service routine is completed.

**Operation:** (BS) ← 1

**Example:** Assume an external interrupt has occurred, control has passed to program memory location 3, and PSW bit 4 was zero before the interrupt.

**Operation:** LOC3: JN1 INIT ;JUMP TO ROUTINE 'INIT' IF  
;INTERRUPT INPUT IS ZERO  
INIT: MOV R7,A ;MOVE ACC CONTENTS TO  
;LOCATION 7  
SEL RB1 ;SELECT REG BANK 1  
MOV R7,#0FAH ;MOVE 'FA' HEX TO LOCATION 31

SEL RB0 ;SELECT REG BANK 0  
MOV A,R7 ;RESTORE ACC FROM LOCATION 7  
RETR ;RETURN — RESTORE PC AND PSW

**STOP TCNT Stop Timer/Event-Counter**Encoding: 

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|

 65H

**Description:** This instruction is used to stop both time accumulation and event counting.

**Example:** Disable interrupt, but jump to interrupt routine after eight overflows and stop timer. Count overflows in register 7.

```

START: DIS TCNTI      ;DISABLE TIMER INTERRUPT
      CLR A           ;CLEAR ACC TO ZEROS
      MOV T,A         ;MOVE ZEROS TO TIMER
      MOV R7,A        ;MOVE ZEROS TO REG 7
      STRT T          ;START TIMER
MAIN:  JTF COUNT       ;JUMP TO ROUTINE 'COUNT'
      ;IF TF = 1 AND CLEAR TIMER FLAG
      JMP MAIN        ;CLOSE LOOP
COUNT: INC R7         ;INCREMENT REG 7
      MOV A,R7        ;MOVE REG 7 CONTENTS TO ACC
      JB3 INT         ;JUMP TO ROUTINE 'INT' IF ACC
      ;BIT 3 IS SET (REG 7 = 8)
      JMP MAIN        ;OTHERWISE RETURN TO ROUTINE
      ;MAIN

INT:   STOP TCNT      ;STOP TIMER
      JMP 7H          ;JUMP TO LOCATION 7 (TIMER)
      ;INTERRUPT ROUTINE

```

#### STRT CNT Start Event Counter

**Encoding:** 0 1 0 0 0 1 0 1 45H

**Description:** The test 1 (T1) pin is enabled as the event-counter input and the counter is started. The event-counter register is incremented with each high-to-low transition on the T1 pin.

**Example:** Initialize and start event counter. Assume overflow is desired with first T1 input.

```

STARTC: EN TCNTI      ;ENABLE COUNTER INTERRUPT
      MOV A,#0FFH     ;MOVE 'FF'HEX (ONES) TO ACC
      MOV T,A         ;MOVES ONES TO COUNTER
      STRT CNT        ;ENABLE T1 AS COUNTER
      ;INPUT AND START

```

## STRT T Start Timer

Encoding: 0 1 0 1 0 1 0 1 55H

**Description:** Timer accumulation is initiated in the timer register. The register is incremented every 32 instruction cycles. The prescaler which counts the 32 cycles is cleared but the timer register is not.

**Example:** Initialize and start timer.

```
STARTT: CLR A           ;CLEAR ACC TO ZEROS
        MOV T,A         ;MOVE ZEROS TO TIMER
        EN TCNTI        ;ENABLE TIMER INTERRUPT
        STRT T          ;START TIMER
```

## SWAP A Swap Nibbles within Accumulator

Encoding: 0 1 0 0 0 1 1 1 47H

**Description:** Bits 0-3 of the accumulator are swapped with bits 4-7 of the accumulator.

**Operation:**  $(A_{4-7}) \leftrightarrow (A_{0-3})$

**Example:** Pack bits 0-3 of locations 50-51 into location 50.

```
PCKDIG: MOV R0, #50      ;MOVE '50' DEC TO REG 0
        MOV R1, #51      ;MOVE '51' DEC TO REG 1
        XCHD A,@R0       ;EXCHANGE BITS 0-3 OF ACC
                           ;AND LOCATION 50
        SWAP A           ;SWAP BITS 0-3 AND 4-7 OF ACC
        XCHD A,@R1       ;EXCHANGE BITS 0-3 OF ACC AND
                           ;LOCATION 51
        MOV @R0,A        ;MOVE CONTENTS OF ACC TO
                           ;LOCATION 50
```

## XCH A,R<sub>r</sub> Exchange Accumulator-Register Contents

Encoding: 0 0 1 0 1 r r r 28H-2FH

**Description:** The contents of the accumulator and the contents of working register 'r' are exchanged.

**Operation:**  $(A) \leftrightarrow (R_r)$   $r = 0-7$

**Example:** Move PSW contents to Reg 7 without losing accumulator contents.

```
XCHAR7: XCH A,R7        ;EXCHANGE CONTENTS OF REG 7
                           ;AND ACC
        MOV A, PSW      ;MOVE PSW CONTENTS TO ACC
        XCH A,R7        ;EXCHANGE CONTENTS OF REG 7
                           ;AND ACC AGAIN
```



**XCH A,@R<sub>i</sub> Exchange Accumulator and Data Memory Contents****Encoding:** 0 0 1 0 0 0 0 i 20H-21H**Description:** The contents of the accumulator and the contents of the resident data memory location addressed by bits 0-5\*\* of register 'i' are exchanged. Register 'i' contents are unaffected.**Operation:** (A)  $\rightleftharpoons$  ((R<sub>i</sub>)) i = 0-1**Example:** Decrement contents of location 52.

```

DEC52: MOV R0,#52      ;MOVE '52' DEC TO ADDRESS REG 0
        XCH A,@R0      ;EXCHANGE CONTENTS OF ACC
                        ;AND LOCATION 52
        DEC A           ;DECREMENT ACC CONTENTS
        XCH A,@R0      ;EXCHANGE CONTENTS OF ACC
                        ;AND LOCATION 52 AGAIN

```

**XCHD A,@R<sub>i</sub> Exchange Accumulator and Data Memory 4-Bit Data****Encoding:** 0 0 1 1 0 0 0 i 30H-31H**Description:** This instruction exchanges bits 0-3 of the accumulator with bits 0-3 of the data memory location addressed by bits 0-5\*\* of register 'i'. Bits 4-7 of the accumulator, bits 4-7 of the data memory location, and the contents of register 'i' are unaffected.**Operation:** (A<sub>0-3</sub>)  $\rightleftharpoons$  ((R<sub>i</sub>0-3)) i = 0-1**Example:** Assume program counter contents have been stacked in locations 22-23.

```

XCHNIB: MOV R0,#23      ;MOVE '23' DEC TO REG 0
        CLR A           ;CLEAR ACC TO ZEROS
        XCHD A,@R0      ;EXCHANGE BITS 0-3 OF ACC AND
                        ;LOCATION 23 (BTS 8-11 OF PC ARE
                        ;ZEROED, ADDRESS REFERS
                        ;TO PAGE 0)

```

**XRL A,R<sub>r</sub> Logical XOR Accumulator With Register Mask****Encoding:** 1 1 0 1 1 r r r D8H-DFH**Description:** Data in the accumulator is EXCLUSIVE ORed with the mask contained in working register 'r'.**Operation:** (A)  $\leftarrow$  (A) XOR (R<sub>r</sub>) r = 0-7**Example:** XORREG: XRL A,R5 ;'XOR' ACC CONTENTS WITH  
;MASK IN REG 5

\*\* 0-5 in 8048AH/8748H  
 0-6 in 8049AH/8749H  
 0-7 in 8050AH

**XRL A,@R<sub>i</sub> Logical XOR Accumulator With Memory Mask**

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 0 | 0 | 0 | i |
|---|---|---|---|---|---|---|---|

 D0H-D1H

**Description:** Data in the accumulator is EXCLUSIVE ORed with the mask contained in the data memory location addressed by register 'i', bits 0-5.\*\*

**Operation:**  $(A) \leftarrow (A) \text{ XOR } ((R_i))$  i = 0-1

**Example:** XORDM: MOV R1,#20H ;MOVE '20' HEX TO REG 1  
                   XRL A,@R1 ;'XOR' ACC CONTENTS WITH MASK  
                               ;IN LOCATION 32

**XRL A,#data Logical XOR Accumulator With Immediate Mask**

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|

|                |                |                |                |                |                |                |                |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| d <sub>7</sub> | d <sub>6</sub> | d <sub>5</sub> | d <sub>4</sub> | d <sub>3</sub> | d <sub>2</sub> | d <sub>1</sub> | d <sub>0</sub> |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|

 D3H

**Description:** This is a 2-cycle instruction. Data in the accumulator is EXCLUSIVE ORed with an immediately-specified mask.

**Operation:**  $(A) \leftarrow (A) \text{ XOR data}$

**Example:** XORID: XOR A,#HEXTEN ;XOR CONTENTS OF ACC WITH MASK  
                                       ;EQUAL VALUE OF SYMBOL 'HEXTEN'

\*\* 0-5 in 8048AH/8748H  
 0-6 in 8049AH/8749H  
 0-7 in 8050AH



---





# 8243 MCS®-48 INPUT/OUTPUT EXPANDER

■ 0° C to 70° C Operation

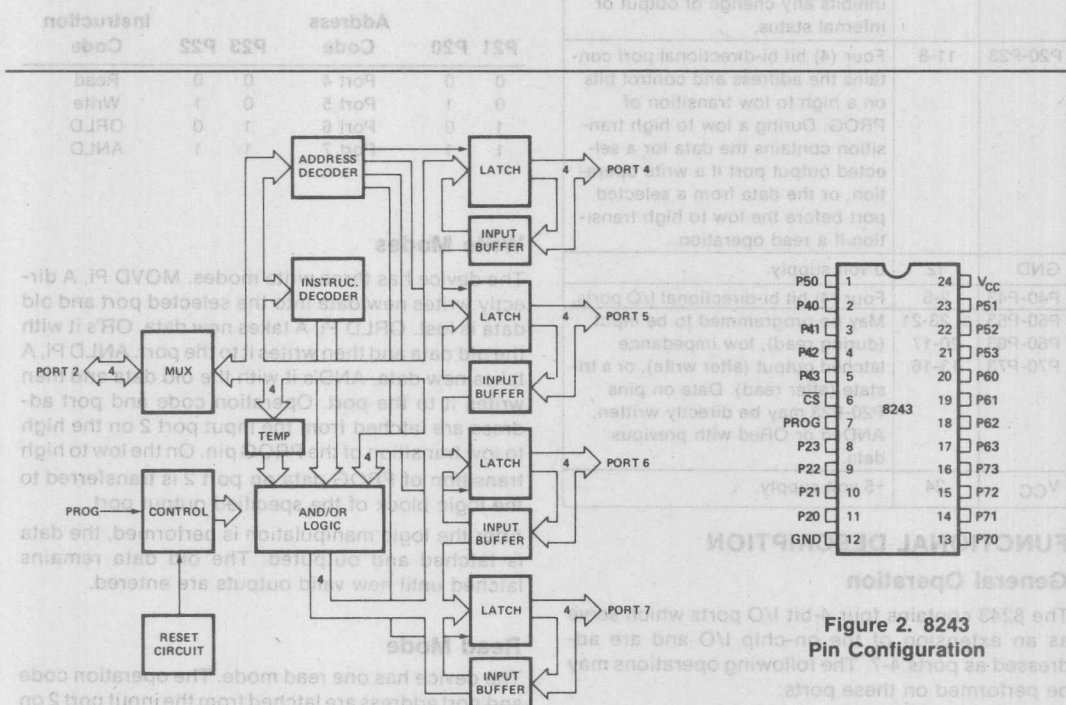


Figure 1. 8243 Block Diagram

Figure 2. 8243 Pin Configuration

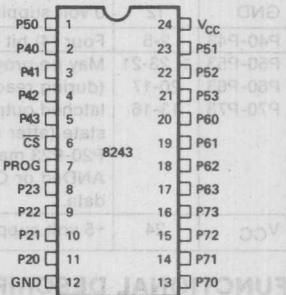


Table 1. Pin Description

| Symbol          | Pin No.  | Function   |
|-----------------|----------|--|
| PROG            | 7        | Clock-Input. A high to low transition on PROG signifies that address and control are available on P20-P23, and a low to high transition signifies that data is available on P20-P23.   |
| $\overline{CS}$ | 6        | Chip Select Input. A high on CS inhibits any change of output or internal status.  |
| P20-P23         | 11-8     | Four (4) bit bi-directional port contains the address and control bits on a high to low transition of PROG. During a low to high transition contains the data for a selected output port if a write operation, or the data from a selected port before the low to high transition if a read operation. |
| GND             | 12       | 0 volt supply.   |
| P40-P43         | 2-5      | Four (4) bit bi-directional I/O ports.   |
| P50-P53         | 1, 23-21 | May be programmed to be input (during read), low impedance latched output (after write), or a tri-state (after read). Data on pins P20-P23 may be directly written, ANDed or ORed with previous data.  |
| P60-P63         | 20-17    |  |
| P70-P73         | 13-16    |  |
| V <sub>CC</sub> | 24       | +5 volt supply.  |

## FUNCTIONAL DESCRIPTION

### General Operation

The 8243 contains four 4-bit I/O ports which serve as an extension of the on-chip I/O and are addressed as ports 4-7. The following operations may be performed on these ports:

- Transfer Accumulator to Port.
- Transfer Port to Accumulator.
- AND Accumulator to Port.
- OR Accumulator to Port.

All communication between the 8048 and the 8243 occurs over Port 2 (P20-P23) with timing provided by an output pulse on the PROG pin of the processor. Each transfer consists of two 4-bit nibbles:

The first containing the "op code" and port address and the second containing the actual 4-bits of data. A high to low transition of the PROG line indicates that address is present while a low to high transition indicates the presence of data. Additional 8243's may be added to the 4-bit bus and chip selected using additional output lines from the 8048/8748/8035.

### Power On Initialization

Initial application of power to the device forces input/output ports 4, 5, 6, and 7 to the tri-state and port 2 to the input mode. The PROG pin may be either high or low when power is applied. The first high to low transition of PROG causes device to exit power on mode. The power on sequence is initiated if VCC drops below 1V.

| Address |     | Code   | Instruction |     | Code  |
|---------|-----|--------|-------------|-----|-------|
| P21     | P20 |        | P23         | P22 |       |
| 0       | 0   | Port 4 | 0           | 0   | Read  |
| 0       | 1   | Port 5 | 0           | 1   | Write |
| 1       | 0   | Port 6 | 1           | 0   | ORLD  |
| 1       | 1   | Port 7 | 1           | 1   | ANLD  |

### Write Modes

The device has three write modes. MOVD Pi, A directly writes new data into the selected port and old data is lost. ORLD Pi, A takes new data, OR's it with the old data and then writes it to the port. ANLD Pi, A takes new data, AND's it with the old data and then writes it to the port. Operation code and port address are latched from the input port 2 on the high to low transition of the PROG pin. On the low to high transition of PROG data on port 2 is transferred to the logic block of the specified output port.

After the logic manipulation is performed, the data is latched and outputted. The old data remains latched until new valid outputs are entered.

### Read Mode

The device has one read mode. The operation code and port address are latched from the input port 2 on the high to low transition of the PROG pin. As soon as the read operation and port address are decoded, the appropriate outputs are tri-stated, and the input buffers switched on. The read operation is terminated by a low to high transition of the PROG pin. The port (4, 5, 6 or 7) that was selected is switched to the tri-stated mode while port 2 is returned to the input mode.

Normally, a port will be in an output (write mode) or input (read mode). If modes are changed during operation, the first read following a write should be ignored; all following reads are valid. This is to allow the external driver on the port to settle after the first read instruction removes the low impedance drive from the 8243 output. A read of any port will leave that port in a high impedance state.

**ABSOLUTE MAXIMUM RATINGS\***

Ambient Temperature Under Bias ..... 0°C to 70°C  
 Storage Temperature ..... -65°C to +150°C  
 Voltage on Any Pin  
   With Respect to Ground ..... -0.5 V to +7V  
 Power Dissipation ..... 1 Watt

\*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

**D.C. CHARACTERISTICS** ( $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ,  $V_{CC} = 5V \pm 10\%$ )

| Symbol           | Parameter                                  | Min  | Typ | Max                  | Units   | Test Conditions                         |
|------------------|--|------|-----|----------------------|---------|---|
| V <sub>IL</sub>  | Input Low Voltage                          | -0.5 |     | 0.8                  | V       |   |
| V <sub>IH</sub>  | Input High Voltage                         | 2.0  |     | V <sub>CC</sub> +0.5 | V       |   |
| V <sub>OL1</sub> | Output Low Voltage Ports 4-7               |      |     | 0.45                 | V       | I <sub>OL</sub> = 4.5 mA*               |
| V <sub>OL2</sub> | Output Low Voltage Port 7                  |      |     | 1                    | V       | I <sub>OL</sub> = 20 mA                 |
| V <sub>OH1</sub> | Output High Voltage Ports 4-7              | 2.4  |     |                      | V       | I <sub>OH</sub> = 240 $\mu$ A           |
| I <sub>IL1</sub> | Input Leakage Ports 4-7                    | -10  |     | 20                   | $\mu$ A | V <sub>in</sub> = V <sub>CC</sub> to 0V |
| I <sub>IL2</sub> | Input Leakage Port 2, CS, PROG             | -10  |     | 10                   | $\mu$ A | V <sub>in</sub> = V <sub>CC</sub> to 0V |
| V <sub>OL3</sub> | Output Low Voltage Port 2                  |      |     | 0.45                 | V       | I <sub>OL</sub> = 0.6 mA                |
| I <sub>CC</sub>  | V <sub>CC</sub> Supply Current             |      | 10  | 20                   | mA      | Note 1                                  |
| V <sub>OH2</sub> | Output Voltage Port 2                      | 2.4  |     |                      |         | I <sub>OH</sub> = 100 $\mu$ A           |
| I <sub>OL</sub>  | Sum of all I <sub>OL</sub> from 16 Outputs |      |     | 72                   | mA      | 4.5 mA Each Pin                         |

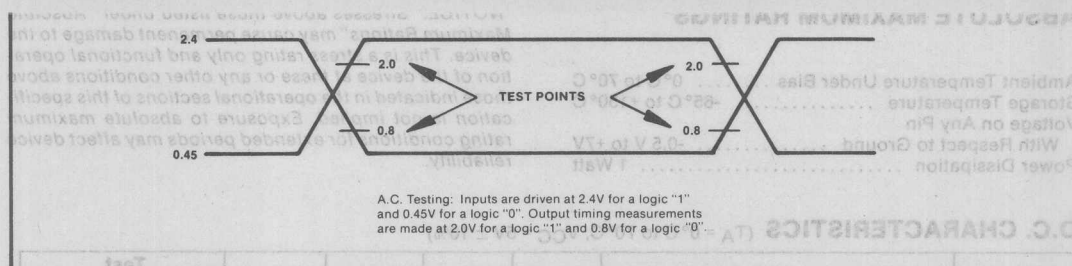
\*See following graph for additional sink current capability

**A.C. CHARACTERISTICS** ( $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ,  $V_{CC} = 5V \pm 10\%$ )

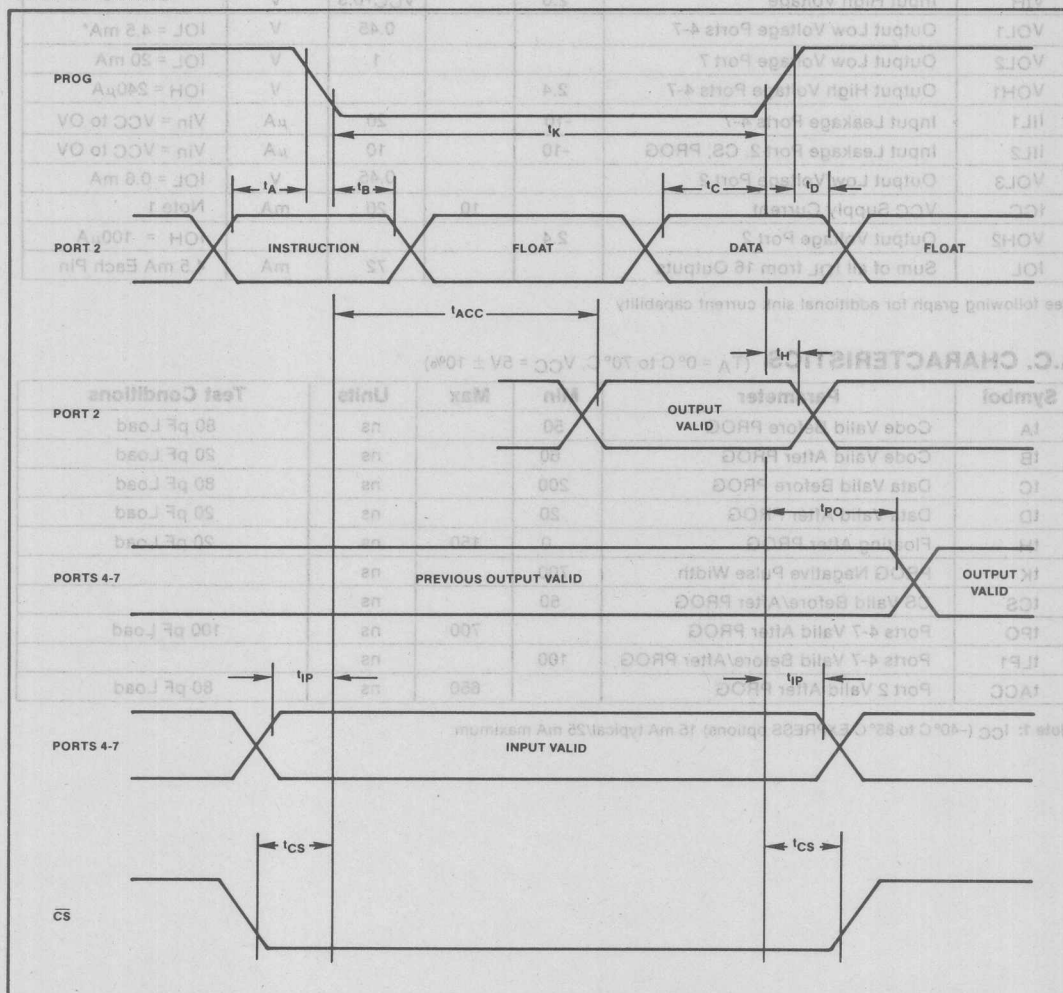
| Symbol           | Parameter                         | Min | Max | Units | Test Conditions |
|------------------|-----------------------------------|-----|-----|-------|-----------------|
| t <sub>A</sub>   | Code Valid Before PROG            | 50  |     | ns    | 80 pF Load      |
| t <sub>B</sub>   | Code Valid After PROG             | 60  |     | ns    | 20 pF Load      |
| t <sub>C</sub>   | Data Valid Before PROG            | 200 |     | ns    | 80 pF Load      |
| t <sub>D</sub>   | Data Valid After PROG             | 20  |     | ns    | 20 pF Load      |
| t <sub>H</sub>   | Floating After PROG               | 0   | 150 | ns    | 20 pF Load      |
| t <sub>K</sub>   | PROG Negative Pulse Width         | 700 |     | ns    |                 |
| t <sub>CS</sub>  | CS Valid Before/After PROG        | 50  |     | ns    |                 |
| t <sub>PO</sub>  | Ports 4-7 Valid After PROG        |     | 700 | ns    | 100 pF Load     |
| t <sub>LP1</sub> | Ports 4-7 Valid Before/After PROG | 100 |     | ns    |                 |
| t <sub>ACC</sub> | Port 2 Valid After PROG           |     | 650 | ns    | 80 pF Load      |

Note 1: I<sub>CC</sub> (-40°C to 85°C EXPRESS options) 15 mA typical/25 mA maximum.





## WAVEFORMS



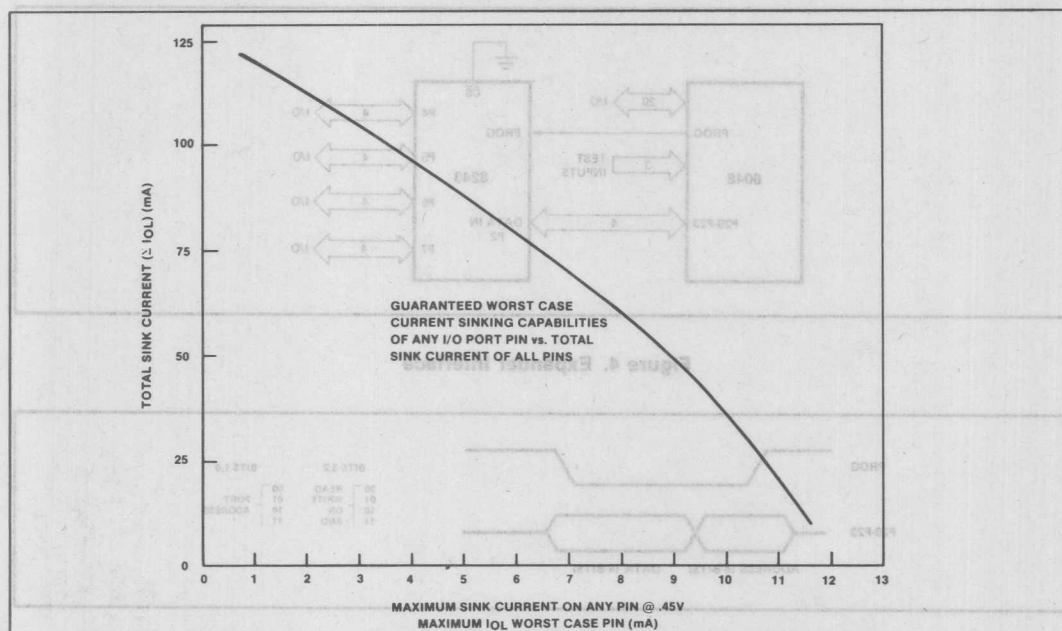


Figure 3

## Sink Capability

The 8243 can sink 5 mA @ .45V on each of its 16 I/O lines simultaneously. If, however, all lines are not sinking simultaneously or all lines are not fully loaded, the drive capability of any individual line increases as is shown by the accompanying curve.

For example, if only 5 of the 16 lines are to sink current at one time, the curve shows that each of those 5 lines is capable of sinking 9 mA @ .45V (if any lines are to sink 9 mA the total IOL must not exceed 45 mA or five 9 mA loads).

Example: How many pins can drive 5 TTL loads (1.6 mA) assuming remaining pins are unloaded?

$$\begin{aligned} \text{IOL} &= 5 \times 1.6 \text{ mA} = 8 \text{ mA} \\ \epsilon\text{IOL} &= 60 \text{ mA from curve} \\ \# \text{ pins} &= 60 \text{ mA} \div 8 \text{ mA/pin} = 7.5 = 7 \end{aligned}$$

In this case, 7 lines can sink 8 mA for a total of 56mA. This leaves 4 mA sink current capability which can be divided in any way among the remaining 8 I/O lines of the 8243.

Example: This example shows how the use of the 20 mA sink capability of Port 7 affects the sinking capability of the other I/O lines.

An 8243 will drive the following loads simultaneously.

- 2 loads—20 mA @ 1V (port 7 only)
- 8 loads—4 mA @ .45V
- 6 loads—3.2 mA @ .45V
- Is this within the specified limits?

$$\begin{aligned} \epsilon\text{IOL} &= (2 \times 20) + (8 \times 4) + (6 \times 3.2) = 91.2 \text{ mA} \\ \text{From the curve: for IOL} &= 4 \text{ mA, } \epsilon\text{IOL} \approx 93 \text{ mA} \\ \text{Since } 91.2 \text{ mA} &< 93 \text{ mA the loads are within specified limits.} \end{aligned}$$

Although the 20 mA @ 1V loads are used in calculating  $\epsilon\text{IOL}$ , it is the largest current required @ .45V which determines the maximum allowable  $\epsilon\text{IOL}$ .

**NOTE:** A10 to 50K  $\Omega$  pullup resistor to +5V should be added to 8243 outputs when driving to 5V CMOS directly.

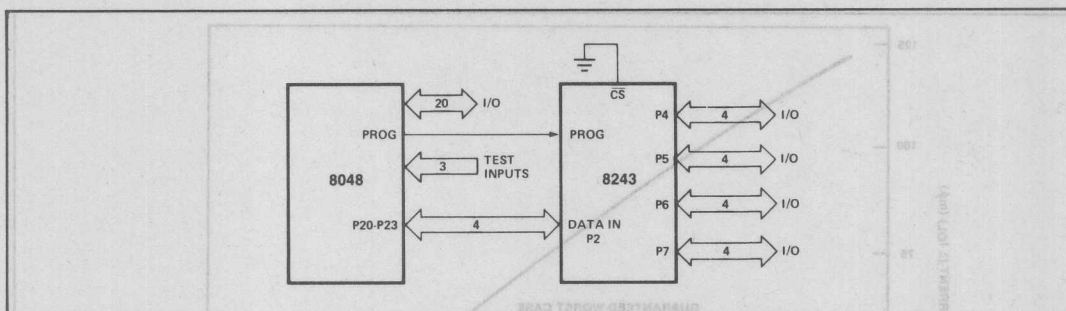


Figure 4. Expander Interface

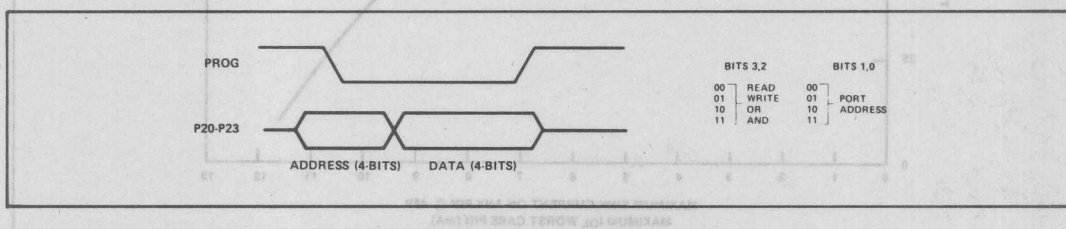


Figure 5. Output Expander Timing

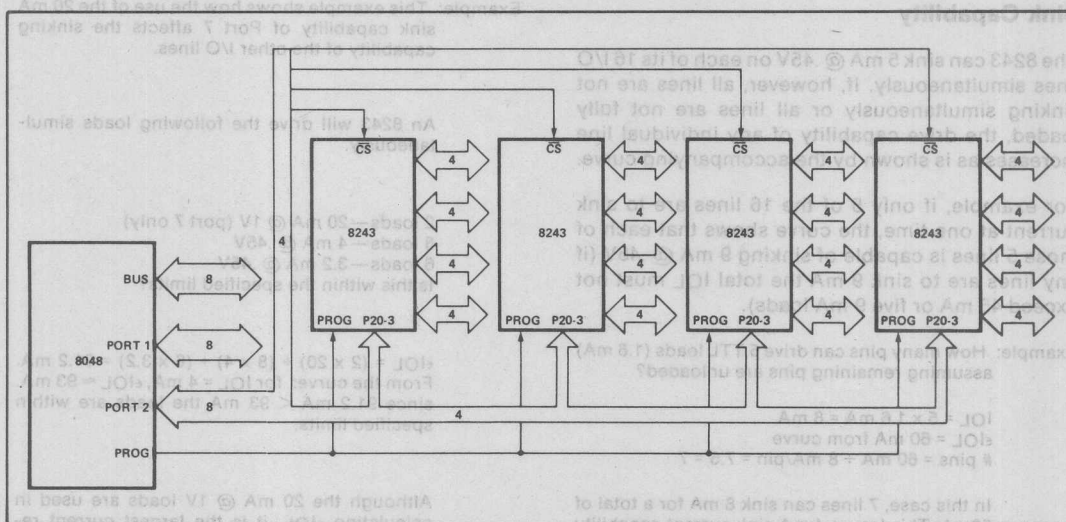


Figure 6. Using Multiple 8243's

# 8048AH/8035AHL/8049AH 8039AHL/8050AH/8040AHL HMOS SINGLE-COMPONENT 8-BIT MICROCOMPUTER

- High Performance HMOS II
- Interval Timer/Event Counter
- Two Single Level Interrupts
- Single 5-Volt Supply
- Over 96 Instructions; 90% Single Byte

- Reduced Power Consumption
- Compatible with 8080/8085 Peripherals
- Easily Expandable Memory and I/O
- Up to 1.36  $\mu$ Sec Instruction Cycle
- All Instructions 1 or 2 cycles

The Intel MCS®-48 family are totally self-sufficient, 8-bit parallel computers fabricated on single silicon chips using Intel's advanced N-channel silicon gate HMOS process.

The family contains 27 I/O lines, an 8-bit timer/counter, and on-board oscillator/clock circuits. For systems that require extra capability, the family can be expanded using MCS®-80/MCS®-85 peripherals.

To minimize development problems and provide maximum flexibility, a logically and functionally pin-compatible version of the ROM devices with UV-erasable user-programmable EPROM program memory is available with minor differences.

These microcomputers are designed to be efficient controllers as well as arithmetic processors. They have extensive bit handling capability as well as facilities for both binary and BCD arithmetic. Efficient use of program memory results from an instruction set consisting mostly of single byte instructions and no instructions over 2 bytes in length.

| Device  | Internal Memory   |                    | RAM Standby |
|---------|-------------------|--------------------|-------------|
| 8050AH  | 4K $\times$ 8 ROM | 256 $\times$ 8 RAM | yes         |
| 8049AH  | 2K $\times$ 8 ROM | 128 $\times$ 8 RAM | yes         |
| 8048AH  | 1K $\times$ 8 ROM | 64 $\times$ 8 RAM  | yes         |
| 8040AHL | none              | 256 $\times$ 8 RAM | yes         |
| 8039AHL | none              | 128 $\times$ 8 RAM | yes         |
| 8035AHL | none              | 64 $\times$ 8 RAM  | yes         |

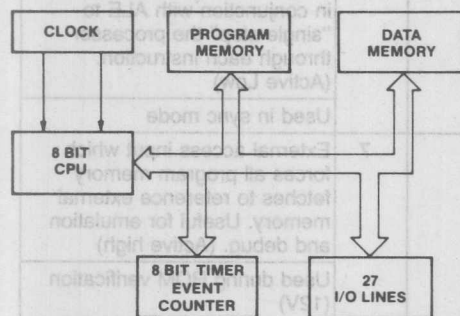


Figure 1.  
Block Diagram

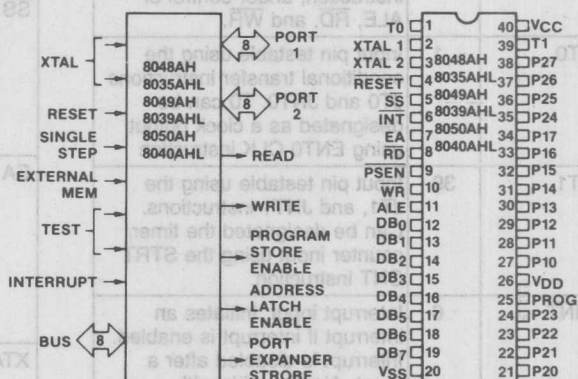


Figure 2.  
Logic Symbol

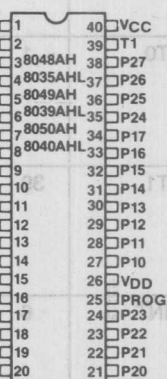


Figure 3.  
Pin Configuration



**Table 1. Pin Description**

| Symbol         | Pin No.        | Function  | Symbol | Pin No. | Function  |
|----------------|----------------|---|--------|---------|---|
| VSS            | 20             | Circuit GND potential   | RD     | 8       | Output strobe activated during a BUS read. Can be used to enable data onto the bus from an external device.<br><br>Used as a read strobe to external data memory. (Active low)          |
| VDD            | 26             | +5V during normal operation.<br><br>Low power standby pin.  | RESET  | 4       | Input which is used to initialize the processor. (Active low) (Non TTL V <sub>IH</sub> )<br><br>Used during power down.<br><br>Used during ROM verification.                            |
| VCC            | 40             | Main power supply; +5V during operation.  | WR     | 10      | Output strobe during a bus write. (Active low)<br><br>Used as write strobe to external data memory.   |
| PROG           | 25             | Output strobe for 8243 I/O expander.  | ALE    | 11      | Address latch enable. This signal occurs once during each cycle and is useful as a clock output.<br><br>The negative edge of ALE strobes address into external data and program memory. |
| P10-P17 Port 1 | 27-34          | 8-bit quasi-bidirectional port.   | PSEN   | 9       | Program store enable. This output occurs only during a fetch to external program memory. (Active low)   |
| P20-P23 Port 2 | 21-24<br>35-38 | 8-bit quasi-bidirectional port. P20-P23 contain the four high order program counter bits during an external program memory fetch and serve as a 4-bit I/O expander bus for 8243.  | SS     | 5       | Single step input can be used in conjunction with ALE to "single step" the processor through each instruction. (Active Low)<br><br>Used in sync mode                                    |
| DB0-DB7 BUS    | 12-19          | True bidirectional port which can be written or read synchronously using the RD, WR strobes. The port can also be statically latched.<br><br>Contains the 8 low order program counter bits during an external program memory fetch, and receives the addressed instruction under the control of PSEN. Also contains the address and data during an external RAM data store instruction, under control of ALE, RD, and WR. | EA     | 7       | External access input which forces all program memory fetches to reference external memory. Useful for emulation and debug. (Active high)<br><br>Used during ROM verification (12V)     |
| T0             | 1              | Input pin testable using the conditional transfer instructions JT0 and JNT0. T0 can be designated as a clock output using ENT0 CLK instruction  | XTAL1  | 2       | One side of crystal input for internal oscillator. Also input for external source. (Non TTL V <sub>IH</sub> )   |
| T1             | 39             | Input pin testable using the JT1, and JNT1 instructions. Can be designated the timer/counter input using the STRT CNT instruction.  | XTAL2  | 3       | Other side of crystal input.  |
| INT            | 6              | Interrupt input. Initiates an interrupt if interrupt is enabled. Interrupt is disabled after a reset. Also testable with conditional jump instruction. (Active low) interrupt must remain low for at least 3 machine cycles for proper operation.   |        |         |   |

Table 2. Instruction Set

| Accumulator    |                               |       |        |
|----------------|-------------------------------|-------|--------|
| Mnemonic       | Description                   | Bytes | Cycles |
| ADD A, R       | Add register to A             | 1     | 1      |
| ADD A, @R      | Add data memory to A          | 1     | 1      |
| ADD A, # data  | Add immediate to A            | 2     | 2      |
| ADDC A, R      | Add register with carry       | 1     | 1      |
| ADDC A, @R     | Add data memory with carry    | 1     | 1      |
| ADDC A, # data | Add immediate with carry      | 2     | 2      |
| ANL A, R       | And register to A             | 1     | 1      |
| ANL A, @R      | And data memory to A          | 1     | 1      |
| ANL A, # data  | And immediate to A            | 2     | 2      |
| ORL A, R       | Or register to A              | 1     | 1      |
| ORL A, @R      | Or data memory to A           | 1     | 1      |
| ORL A, # data  | Or immediate to A             | 2     | 2      |
| XRL A, R       | Exclusive or register to A    | 1     | 1      |
| XRL A, @R      | Exclusive or data memory to A | 1     | 1      |
| XRL A, # data  | Exclusive or immediate to A   | 2     | 2      |
| INC A          | Increment A                   | 1     | 1      |
| DEC A          | Decrement A                   | 1     | 1      |
| CLR A          | Clear A                       | 1     | 1      |
| CPL A          | Complement A                  | 1     | 1      |
| DA A           | Decimal adjust A              | 1     | 1      |
| SWAP A         | Swap nibbles of A             | 1     | 1      |
| RL A           | Rotate A left                 | 1     | 1      |
| RLC A          | Rotate A left through carry   | 1     | 1      |
| RR A           | Rotate A right                | 1     | 1      |
| RRC A          | Rotate A right through carry  | 1     | 1      |

| Input/Output    |                           |       |        |
|-----------------|---------------------------|-------|--------|
| Mnemonic        | Description               | Bytes | Cycles |
| IN A, P         | Input port to A           | 1     | 2      |
| OUTL P, A       | Output A to port          | 1     | 2      |
| ANL P, # data   | And immediate to port     | 2     | 2      |
| ORL P, # data   | Or immediate to port      | 2     | 2      |
| INS A, BUS      | Input BUS to A            | 1     | 2      |
| OUTL BUS, A     | Output A to BUS           | 1     | 2      |
| ANL BUS, # data | And immediate to BUS      | 2     | 2      |
| ORL BUS, # data | Or immediate to BUS       | 2     | 2      |
| MOVD A, P       | Input expander port to A  | 1     | 2      |
| MOVD P, A       | Output A to expander port | 1     | 2      |
| ANLD P, A       | And A to expander port    | 1     | 2      |
| ORLD P, A       | Or A to expander port     | 1     | 2      |

| Registers |                       |       |        |
|-----------|-----------------------|-------|--------|
| Mnemonic  | Description           | Bytes | Cycles |
| INC R     | Increment register    | 1     | 1      |
| INC @R    | Increment data memory | 1     | 1      |
| DEC R     | Decrement register    | 1     | 1      |

| Branch       |                             |       |        |
|--------------|-----------------------------|-------|--------|
| Mnemonic     | Description                 | Bytes | Cycles |
| JMP addr     | Jump unconditional          | 2     | 2      |
| JMPP @A      | Jump indirect               | 1     | 2      |
| DJNZ R, addr | Decrement register and skip | 2     | 2      |
| JC addr      | Jump on carry = 1           | 2     | 2      |
| JNC addr     | Jump on carry = 0           | 2     | 2      |
| JZ addr      | Jump on A zero              | 2     | 2      |
| JNZ addr     | Jump on A not zero          | 2     | 2      |
| JT0 addr     | Jump on T0 = 1              | 2     | 2      |
| JNT0 addr    | Jump on T0 = 0              | 2     | 2      |
| JT1 addr     | Jump on T1 = 1              | 2     | 2      |
| JNT1 addr    | Jump on T1 = 0              | 2     | 2      |
| JF0 addr     | Jump on F0 = 1              | 2     | 2      |
| JF1 addr     | Jump on F1 = 1              | 2     | 2      |
| JTF addr     | Jump on timer flag          | 2     | 2      |
| JNI addr     | Jump on INT = 0             | 2     | 2      |
| JBb addr     | Jump on accumulator bit     | 2     | 2      |

| Subroutine |                           |       |        |
|------------|---------------------------|-------|--------|
| Mnemonic   | Description               | Bytes | Cycles |
| CALL addr  | Jump to subroutine        | 2     | 2      |
| RET        | Return                    | 1     | 2      |
| RETR       | Return and restore status | 1     | 2      |

| Flags    |                   |       |        |
|----------|-------------------|-------|--------|
| Mnemonic | Description       | Bytes | Cycles |
| CLR C    | Clear carry       | 1     | 1      |
| CPL C    | Complement carry  | 1     | 1      |
| CLR F0   | Clear flag 0      | 1     | 1      |
| CPL F0   | Complement flag 0 | 1     | 1      |
| CLR F1   | Clear flag 1      | 1     | 1      |
| CPL F1   | Complement flag 1 | 1     | 1      |

## Data Moves

| Mnemonic       | Description                       | Bytes | Cycles |
|----------------|-----------------------------------|-------|--------|
| MOV A, R       | Move register to A                | 1     | 1      |
| MOV A, @R      | Move data memory to A             | 1     | 1      |
| MOV A, # data  | Move immediate to A               | 2     | 2      |
| MOV R, A       | Move A to register                | 1     | 1      |
| MOV @R, A      | Move A to data memory             | 1     | 1      |
| MOV R, # data  | Move immediate to register        | 2     | 2      |
| MOV @R, # data | Move immediate to data memory     | 2     | 2      |
| MOV A, PSW     | Move PSW to A                     | 1     | 1      |
| MOV PSW, A     | Move A to PSW                     | 1     | 1      |
| XCH A, R       | Exchange A and register           | 1     | 1      |
| XCHD A, @R     | Exchange nibble of A and register | 1     | 1      |
| MOVB A, @R     | Move external data memory to A    | 1     | 2      |
| MOVB @R, A     | Move A to external data memory    | 1     | 2      |
| MOVP A, @A     | Move to A from current page       | 1     | 2      |
| MOVP3 A, @     | Move to A from page 3             | 1     | 2      |

## Timer/Counter

| Mnemonic  | Description                     | Bytes | Cycles |
|-----------|---------------------------------|-------|--------|
| MOV A, T  | Read timer/counter              | 1     | 1      |
| MOV T, A  | Load timer/counter              | 1     | 1      |
| STRT T    | Start timer                     | 1     | 1      |
| STRT CNT  | Start timer                     | 1     | 1      |
| STOP TCNT | Stop timer/counter              | 1     | 1      |
| EN TCNTI  | Enable timer/counter interrupt  | 1     | 1      |
| DIS TCNTI | Disable timer/counter interrupt | 1     | 1      |

## Control

| Mnemonic | Description                | Bytes | Cycles |
|----------|----------------------------|-------|--------|
| EN I     | Enable external interrupt  | 1     | 1      |
| DIS I    | Disable external interrupt | 1     | 1      |
| SEL RB0  | Select register bank 0     | 1     | 1      |
| SEL RB1  | Select register bank 1     | 1     | 1      |
| SEL MB0  | Select memory bank 0       | 1     | 1      |
| SEL MB1  | Select memory bank 1       | 1     | 1      |
| ENT0 CLK | Enable clock output on T0  | 1     | 1      |

| Mnemonic | Description           | Bytes | Cycles |
|----------|-----------------------|-------|--------|
| NOP      | No operation          | 1     | 1      |
| IDL      | Select Idle Operation | 1     | 1      |

| Mnemonic | Description               | Bytes | Cycles |
|----------|---------------------------|-------|--------|
| CALL     | Jump to subroutine        | 2     | 2      |
| RET      | Return                    | 1     | 2      |
| RETR     | Return and restore status | 1     | 2      |

| Mnemonic | Description       | Bytes | Cycles |
|----------|-------------------|-------|--------|
| CLR C    | Clear carry       | 1     | 1      |
| CPL C    | Complement carry  | 1     | 1      |
| CLR F0   | Clear flag 0      | 1     | 1      |
| CPL F0   | Complement flag 0 | 1     | 1      |
| CLR F1   | Clear flag 1      | 1     | 1      |
| CPL F1   | Complement flag 1 | 1     | 1      |

| Mnemonic          | Description               | Bytes | Cycles |
|-------------------|---------------------------|-------|--------|
| IN A, P           | Input port to A           | 1     | 2      |
| OUT A, P          | Output A to port          | 1     | 2      |
| IN A, P, # data   | And immediate to port     | 2     | 2      |
| OR A, P, # data   | Or immediate to port      | 2     | 2      |
| IN A, BUS         | Input BUS to A            | 1     | 2      |
| OUT A, BUS        | Output A to BUS           | 1     | 2      |
| IN A, BUS, # data | And immediate to BUS      | 2     | 2      |
| OR A, BUS, # data | Or immediate to BUS       | 2     | 2      |
| MOV A, P, # data  | Input expander port to A  | 1     | 2      |
| MOV A, P, # data  | Output A to expander port | 1     | 2      |
| IN A, P, # data   | And A to expander port    | 1     | 2      |
| OR A, P, # data   | Or A to expander port     | 1     | 2      |

**ABSOLUTE MAXIMUM RATINGS\***

Ambient Temperature Under Bias . . . 0°C to 70°C  
Storage Temperature . . . . . -65°C to +150°C  
Voltage On Any Pin With Respect  
to Ground . . . . . -0.5V to +7V  
Power Dissipation . . . . . 1.5 Watt

*\*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of device at these or any other conditions above those indicated in the operational sections of this specification is not implied.*

**D.C. CHARACTERISTICS:** ( $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ;  $V_{CC} = V_{DD} = 5V \pm 10\%$ ;  $V_{SS} = 0V$ )

| Symbol    | Parameter   | Limits |     |          | Unit | Test Conditions                    | Device |
|-----------|---|--------|-----|----------|------|------------------------------------|--------|
|           |   | Min    | Typ | Max      |      |                                    |        |
| $V_{IL}$  | Input Low Voltage (All Except RESET, X1, X2)        | - .5   |     | .8       | V    |                                    | All    |
| $V_{IL1}$ | Input Low Voltage (RESET, X1, X2)                   | - .5   |     | .6       | V    |                                    | All    |
| $V_{IH}$  | Input High Voltage (All Except XTAL1, XTAL2, RESET) | 2.0    |     | $V_{CC}$ | V    |                                    | All    |
| $V_{IH1}$ | Input High Voltage (X1, X2, RESET)                  | 3.8    |     | $V_{CC}$ | V    |                                    | All    |
| $V_{OL}$  | Output Low Voltage (BUS)                            |        |     | .45      | V    | $I_{OL} = 2.0\text{ mA}$           | All    |
| $V_{OL1}$ | Output Low Voltage (RD, WR, PSEN, ALE)              |        |     | .45      | V    | $I_{OL} = 1.8\text{ mA}$           | All    |
| $V_{OL2}$ | Output Low Voltage (PROG)                           |        |     | .45      | V    | $I_{OL} = 1.0\text{ mA}$           | All    |
| $V_{OL3}$ | Output Low Voltage (All Other Outputs)              |        |     | .45      | V    | $I_{OL} = 1.6\text{ mA}$           | All    |
| $V_{OH}$  | Output High Voltage (BUS)                           | 2.4    |     |          | V    | $I_{OH} = -400\text{ }\mu\text{A}$ | All    |
| $V_{OH1}$ | Output High Voltage (RD, WR, PSEN, ALE)             | 2.4    |     |          | V    | $I_{OH} = -100\text{ }\mu\text{A}$ | All    |
| $V_{OH2}$ | Output High Voltage (All Other Outputs)             | 2.4    |     |          | V    | $I_{OH} = -40\text{ }\mu\text{A}$  | All    |



**D.C. CHARACTERISTICS:** ( $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ;  $V_{CC} = V_{DD} = 5\text{V} \pm 10\%$ ;  $V_{SS} = 0\text{V}$ ) (Continued)

| Symbol            | Parameter   | Limits |     |          | Unit          | Test Conditions                        | Device            |
|-------------------|---|--------|-----|----------|---------------|--|-------------------|
|                   |   | Min    | Typ | Max      |               |  |                   |
| $I_{L1}$          | Leakage Current<br>( $T_1$ , $\overline{\text{INT}}$ )                      |        |     | $\pm 10$ | $\mu\text{A}$ | $V_{SS} \leq V_{IN} \leq V_{CC}$       | All               |
| $I_{LI1}$         | Input Leakage Current<br>(P10-P17, P20-P27,<br>EA, $\overline{\text{SS}}$ ) |        |     | -500     | $\mu\text{A}$ | $V_{SS} + .45 \leq V_{IN} \leq V_{CC}$ | All               |
| $I_{LI2}$         | Input Leakage Current<br>$\overline{\text{RESET}}$                          | 20     |     | 300      | $\mu\text{A}$ | $V_{SS} \leq V_{IN} \leq 3.8\text{V}$  | All               |
| $I_{LO}$          | Leakage Current<br>(BUS, $T_0$ ) (High<br>Impedance State)                  |        |     | $\pm 10$ | $\mu\text{A}$ | $V_{SS} \leq V_{IN} \leq V_{CC}$       | All               |
| $I_{DD}$          | $V_{DD}$ Supply Current<br>(RAM Standby)<br>IIA                             |        | 3   | 5        | mA            |  | 8048AH<br>8035AHL |
|                   |   |        | 4   | 7        | mA            |  | 8049AH<br>8039AHL |
|                   |   |        | 5   | 10       | mA            |  | 8050AH<br>8040AHL |
|                   |   |        |     |          |               |  |                   |
| $I_{DD} + I_{CC}$ | Total Supply Current*<br>IIA  |        | 30  | 65       | mA            |  | 8048AH<br>8035AHL |
|                   |   |        | 35  | 70       | mA            |  | 8049AH<br>8039AHL |
|                   |   |        | 40  | 80       | mA            |  | 8050AH<br>8040AHL |
|                   |   |        |     |          |               |  |                   |
| $V_{DD}$          | RAM Standby Voltage   | 2.2    |     | 5.5      | V             | Standby Mode Reset<br>$\leq V_{IL1}$   | All               |

\* $I_{CC} + I_{DD}$  is measured with all outputs disconnected;  $\overline{\text{SS}}$ ,  $\overline{\text{RESET}}$ , and  $\overline{\text{INT}}$  equal to  $V_{CC}$ ; EA equal to  $V_{SS}$ .

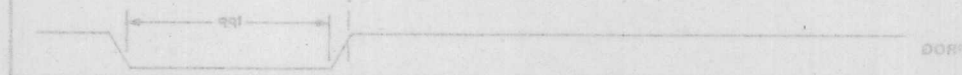
**A.C. CHARACTERISTICS:** ( $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ;  $V_{CC} = V_{DD} = 5V \pm 10\%$ ;  $V_{SS} = 0V$ )

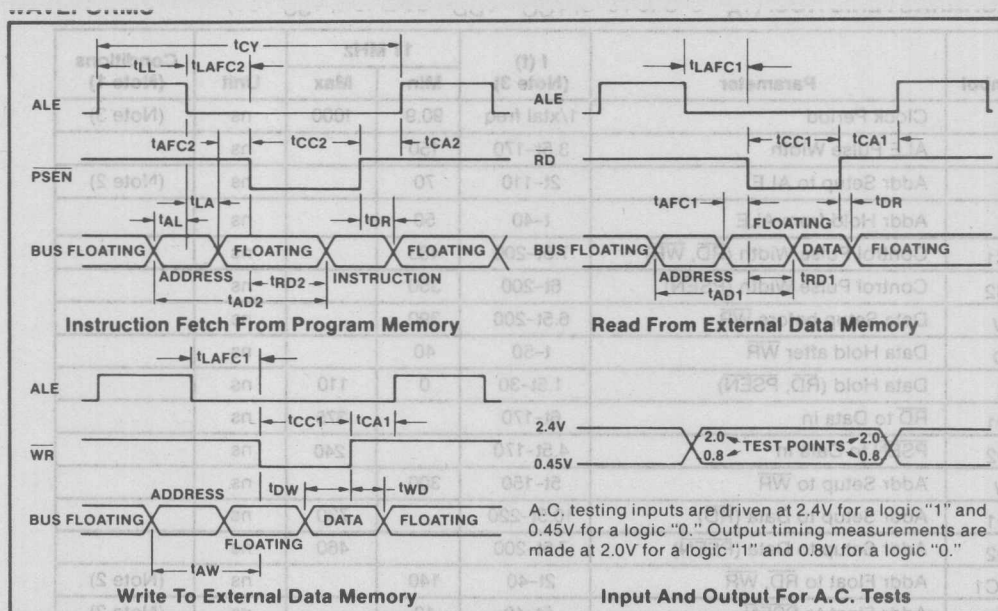
WAVEFORMS

| Symbol             | Parameter  | f (t)<br>(Note 3) | 11 MHz |      | Unit          | Conditions<br>(Note 1) |
|--------------------|--|-------------------|--------|------|---------------|------------------------|
|                    |  |                   | Min    | Max  |               |                        |
| t                  | Clock Period   | 1/xtal freq       | 90.9   | 1000 | ns            | (Note 3)               |
| t <sub>LL</sub>    | ALE Pulse Width  | 3.5t-170          | 150    |      | ns            |                        |
| t <sub>AL</sub>    | Addr Setup to ALE  | 2t-110            | 70     |      | ns            | (Note 2)               |
| t <sub>LA</sub>    | Addr Hold from ALE   | t-40              | 50     |      | ns            |                        |
| t <sub>CC1</sub>   | Control Pulse Width ( $\overline{RD}$ , $\overline{WR}$ )                | 7.5t-200          | 480    |      | ns            |                        |
| t <sub>CC2</sub>   | Control Pulse Width ( $\overline{PSEN}$ )                                | 6t-200            | 350    |      | ns            |                        |
| t <sub>DW</sub>    | Data Setup before $\overline{WR}$  | 6.5t-200          | 390    |      | ns            |                        |
| t <sub>WD</sub>    | Data Hold after $\overline{WR}$  | t-50              | 40     |      | ns            |                        |
| t <sub>DR</sub>    | Data Hold ( $\overline{RD}$ , $\overline{PSEN}$ )                        | 1.5t-30           | 0      | 110  | ns            |                        |
| t <sub>RD1</sub>   | $\overline{RD}$ to Data in   | 6t-170            |        | 375  | ns            |                        |
| t <sub>RD2</sub>   | $\overline{PSEN}$ to Data in   | 4.5t-170          |        | 240  | ns            |                        |
| t <sub>AW</sub>    | Addr Setup to $\overline{WR}$  | 5t-150            | 300    |      | ns            |                        |
| t <sub>AD1</sub>   | Addr Setup to Data ( $\overline{RD}$ )                                   | 10.5t-220         |        | 730  | ns            |                        |
| t <sub>AD2</sub>   | Addr Setup to Data ( $\overline{PSEN}$ )                                 | 7.5t-200          |        | 460  | ns            |                        |
| t <sub>AFC1</sub>  | Addr Float to $\overline{RD}$ , $\overline{WR}$                          | 2t-40             | 140    |      | ns            | (Note 2)               |
| t <sub>AFC2</sub>  | Addr Float to $\overline{PSEN}$  | .5t-40            | 10     |      | ns            | (Note 2)               |
| t <sub>LAFC1</sub> | ALE to Control ( $\overline{RD}$ , $\overline{WR}$ )                     | 3t-75             | 200    |      | ns            |                        |
| t <sub>LAFC2</sub> | ALE to Control ( $\overline{PSEN}$ )                                     | 1.5t-75           | 60     |      | ns            |                        |
| t <sub>CA1</sub>   | Control to ALE ( $\overline{RD}$ , $\overline{WR}$ , $\overline{PROG}$ ) | t-65              | 25     |      | ns            |                        |
| t <sub>CA2</sub>   | Control to ALE ( $\overline{PSEN}$ )                                     | 4t-70             | 290    |      | ns            |                        |
| t <sub>CP</sub>    | Port Control Setup to $\overline{PROG}$                                  | 1.5t-80           | 50     |      | ns            |                        |
| t <sub>PC</sub>    | Port Control Hold to $\overline{PROG}$                                   | 4t-260            | 100    |      | ns            |                        |
| t <sub>PR</sub>    | $\overline{PROG}$ to P2 Input Valid                                      | 8.5t-120          |        | 650  | ns            |                        |
| t <sub>PF</sub>    | Input Data Hold from $\overline{PROG}$                                   | 1.5t              | 0      | 140  | ns            |                        |
| t <sub>DP</sub>    | Output Data Setup  | 6t-290            | 250    |      | ns            |                        |
| t <sub>PD</sub>    | Output Data Hold   | 1.5t-90           | 40     |      | ns            |                        |
| t <sub>PP</sub>    | $\overline{PROG}$ Pulse Width  | 10.5t-250         | 700    |      | ns            |                        |
| t <sub>PL</sub>    | Port 2 I/O Setup to ALE  | 4t-200            | 160    |      | ns            |                        |
| t <sub>LP</sub>    | Port 2 I/O Hold to ALE   | .5t-30            | 15     |      | ns            |                        |
| t <sub>PV</sub>    | Port Output from ALE   | 4.5t-100          |        | 510  | ns            |                        |
| t <sub>0PRR</sub>  | T0 Rep Rate  | 3t                | 270    |      | ns            |                        |
| t <sub>CY</sub>    | Cycle Time   | 15t               | 1.36   | 15.0 | $\mu\text{s}$ |                        |

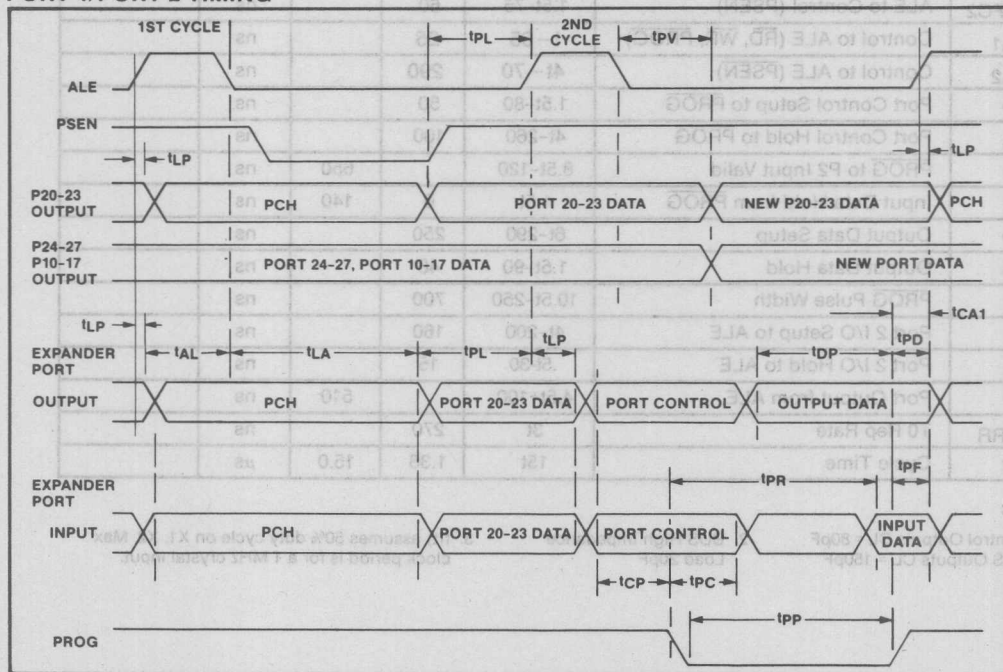
**Notes:**

- Control Outputs  $CL = 80\text{pF}$   
BUS Outputs  $CL = 150\text{pF}$
- BUS High Impedance  
Load  $20\text{pF}$
- f(t) assumes 50% duty cycle on X1, X2. Max clock period is for a 1 MHz crystal input.

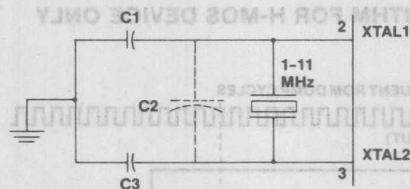




#### PORT 1/PORT 2 TIMING



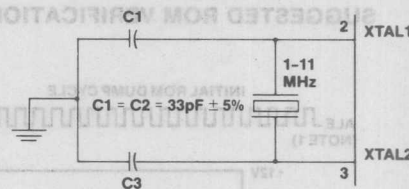
### CRYSTAL OSCILLATOR MODE



$$\begin{aligned} C1 &= 5\text{pF} \pm 1/2\text{pF} + (\text{STRAY} < 5\text{pF}) \\ C2 &= (\text{CRYSTAL} + \text{STRAY}) < 8\text{pF} \\ C3 &= 20\text{pF} \pm 1\text{pF} + (\text{STRAY} < 5\text{pF}) \end{aligned}$$

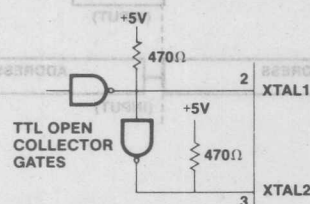
Crystal series resistance should be less than 30Ω at 11 MHz;  
less than 75Ω at 6 MHz; less than 180Ω at 3.6 MHz.

### CERAMIC RESONATOR MODE



$$C1 = C2 = 33\text{pF} \pm 5\%$$

### DRIVING FROM EXTERNAL SOURCE

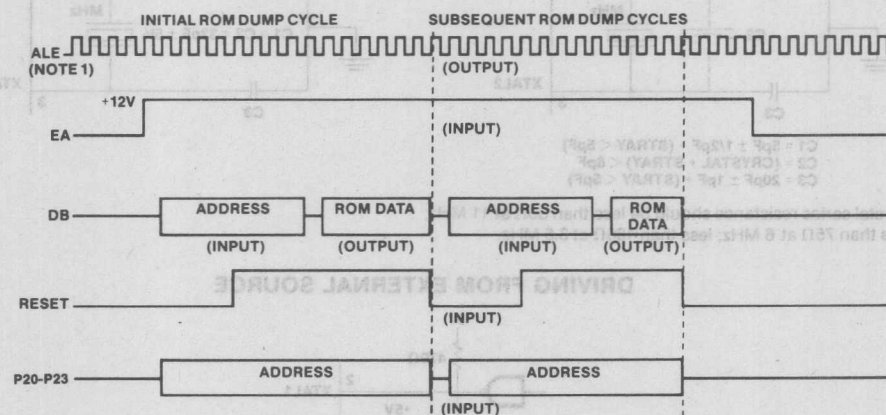


For XTAL1 and XTAL2 define "high" as voltages above 1.6V and "low" as voltages below 1.6V. The duty cycle requirements for externally driving XTAL1 and XTAL2 using the

circuit shown above are as follows: XTAL1 must be high 35-65% of the period and XTAL2 must be high 36-65% of the period. Rise and fall times must be faster than 20 nS.



# SUGGESTED ROM VERIFICATION ALGORITHM FOR H-MOS DEVICE ONLY



VCC = VDD = + 5V  
VSS = 0V

|     | 48H | 49H  | 50H  |
|-----|-----|------|------|
| A10 | 0   | ADDR | ADDR |
| A11 | 0   | 0    | ADDR |

**NOTE:** ALE is function of X1, X2 inputs.

# 8748H/8035H/8749H/8039H HMOS-E SINGLE-COMPONENT 8-BIT MICROCOMPUTER

- High Performance HMOS-E
- Interval Timer/Event Counter
- Two Single Level Interrupts
- Single 5-Volt Supply
- Over 96 Instructions;  
90% Single Byte
- Compatible with 8080/8085  
Peripherals
- Easily Expandable Memory and I/O
- Up to 1.35  $\mu$ Sec Instruction Cycle  
All Instructions 1 or 2 cycles

The Intel 8749H/8039H/8748H/8035H are totally self-sufficient, 8-bit parallel computers fabricated on single silicon chips using Intel's advanced N-channel silicon gate HMOS-E process.

The family contains 27 I/O lines, an 8-bit timer/counter, on-chip RAM and on-board oscillator/clock circuits. For systems that require extra capability, the family can be expanded using MCS<sup>®</sup>-80/MCS<sup>®</sup>-85 peripherals.

These microcomputers are designed to be efficient controllers as well as arithmetic processors. They have extensive bit handling capability as well as facilities for both binary and BCD arithmetic. Efficient use of program memory results from an instruction set consisting mostly of single byte instructions and no instructions over 2 bytes in length.

| Device | Internal Memory          |
|--------|--------------------------|
| 8039H  | none 128 x 8 RAM         |
| 8035H  | none 64 x 8 RAM          |
| 8749H  | 2K x 8 EPROM 128 x 8 RAM |
| 8748H  | 1K x 8 EPROM 64 x 8 RAM  |

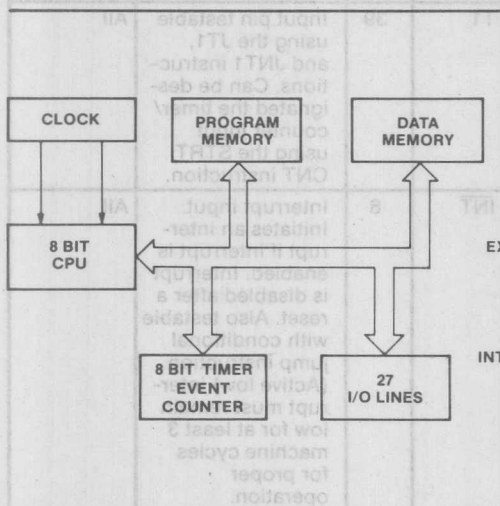


Figure 1.  
Block Diagram

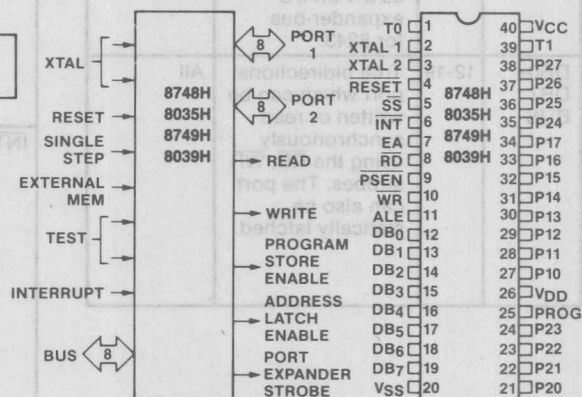


Figure 2.  
Logic Symbol

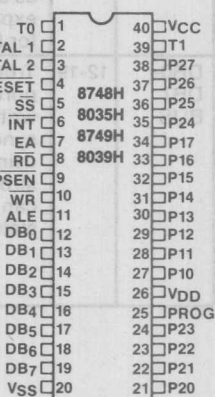


Figure 3.  
Pin Configuration

Table 1. Pin Description

| Symbol                 | Pin No.        | Function   | Device                       | Symbol  | Pin No. | Function   | Device         |
|------------------------|----------------|--|------------------------------|---------|---------|--|----------------|
| V <sub>SS</sub>        | 20             | Circuit GND potential  | All                          | (Con't) |         | Contains the 8 low order program counter bits during an external program memory fetch, and receives the addressed instruction under the control of PSEN. Also contains the address and data during an external RAM data store instruction, under control of ALE, RD, and WR. |                |
| V <sub>DD</sub>        | 26             | +5V during normal operation.   | All                          |         |         |  |                |
|                        |                | Programming power supply (+21V).   | 8748H<br>8749H               |         |         |  |                |
| V <sub>CC</sub>        | 40             | Main power supply; +5V during operation and programming.   | All                          |         |         |  |                |
| PROG                   | 25             | Output strobe for 8243 I/O expander.   | All                          |         |         |  |                |
|                        |                | Program pulse (+18V) input pin during programming.   | 8748H<br>8749H<br>(See Note) | T0      | 1       | Input pin testable using the conditional transfer instructions JT0 and JNT0. T0 can be designated as a clock output using ENT0 CLK instruction   | All            |
| P10-P17 Port 1         | 27-34          | 8-bit quasi-bidirectional port.  | All                          |         |         | Used during programming.   | 8748H<br>8749H |
| P20-P23 P24-P27 Port 2 | 21-24<br>35-38 | 8-bit quasi-bidirectional port. P20-P23 contain the four high order program counter bits during an external program memory fetch and serve as a 4-bit I/O expander bus for 8243. | All                          | T1      | 39      | Input pin testable using the JT1, and JNT1 instructions. Can be designated the timer/counter input using the STRT CNT instruction.   | All            |
| DB0-DB7 BUS            | 12-19          | True bidirectional port which can be written or read synchronously using the RD, WR strobes. The port can also be statically latched.  | All                          | INT     | 6       | Interrupt input. Initiates an interrupt if interrupt is enabled. Interrupt is disabled after a reset. Also testable with conditional jump instruction. (Active low) interrupt must remain low for at least 3 machine cycles for proper operation.                            | All            |

Table 1. Pin Description (Continued)

| Symbol | Pin No. | Function  | Device                |
|--------|---------|---|-----------------------|
| RD     | 8       | Output strobe activated during a BUS read. Can be used to enable data onto the bus from an external device.<br><br>Used as a read strobe to external data memory. (Active low)          | All                   |
| RESET  | 4       | Input which is used to initialize the processor. (Active low) (Non TTL V <sub>IH</sub> )<br><br>Used during programming.  | All<br>8748H<br>8749H |
| WR     | 10      | Output strobe during a bus write. (Active low)<br><br>Used as write strobe to external data memory.   | All                   |
| ALE    | 11      | Address latch enable. This signal occurs once during each cycle and is useful as a clock output.<br><br>The negative edge of ALE strobes address into external data and program memory. | All                   |
| PSEN   | 9       | Program store enable. This output occurs only during a fetch to external program memory. (Active low)   | All                   |
| SS     | 5       | Single step input can be used in conjunction with ALE to "single step" the processor through each instruction.  | All                   |
| EA     | 7       | External access input which forces all program memory fetches to reference external memory. Useful for emulation and debug. (Active high)<br><br>Used during (18V) programming.         | All<br>8748H<br>8749H |
| XTAL1  | 2       | One side of crystal input for internal oscillator. Also input for external source. (Non TTL V <sub>IH</sub> )   | All                   |
| XTAL2  | 3       | Other side of crystal input.  | All                   |

**NOTE:** On the 8749H/8039H, PROG must be clamped to V<sub>CC</sub> when not programming. A diode should be used when using an 8243; otherwise, a direct connection is permissible.



**Table 2. Instruction Set**

| <b>Accumulator</b>  |                               |              |               | <b>Registers</b> |                             |              |               |
|---------------------|-------------------------------|--------------|---------------|------------------|-----------------------------|--------------|---------------|
| <b>Mnemonic</b>     | <b>Description</b>            | <b>Bytes</b> | <b>Cycles</b> | <b>Mnemonic</b>  | <b>Description</b>          | <b>Bytes</b> | <b>Cycles</b> |
| ADD A, R            | Add register to A             | 1            | 1             | INC R            | Increment register          | 1            | 1             |
| ADD A, @R           | Add data memory to A          | 1            | 1             | INC @R           | Increment data memory       | 1            | 1             |
| ADD A, # data       | Add immediate to A            | 2            | 2             | DEC R            | Decrement register          | 1            | 1             |
| ADDC A, R           | Add register with carry       | 1            | 1             |                  |                             |              |               |
| ADDC A, @R          | Add data memory with carry    | 1            | 1             |                  |                             |              |               |
| ADDC A, # data      | Add immediate with carry      | 2            | 2             |                  |                             |              |               |
| ANL A, R            | And register to A             | 1            | 1             |                  |                             |              |               |
| ANL A, @R           | And data memory to A          | 1            | 1             |                  |                             |              |               |
| ANL A, # data       | And immediate to A            | 2            | 2             |                  |                             |              |               |
| ORL A, R            | Or register to A              | 1            | 1             |                  |                             |              |               |
| ORL A, @R           | Or data memory to A           | 1            | 1             |                  |                             |              |               |
| ORL A, # data       | Or immediate to A             | 2            | 2             |                  |                             |              |               |
| XRL A, R            | Exclusive or register to A    | 1            | 1             |                  |                             |              |               |
| XRL A, @R           | Exclusive or data memory to A | 1            | 1             |                  |                             |              |               |
| XRL A, # data       | Exclusive or immediate to A   | 2            | 2             |                  |                             |              |               |
| INC A               | Increment A                   | 1            | 1             |                  |                             |              |               |
| DEC A               | Decrement A                   | 1            | 1             |                  |                             |              |               |
| CLR A               | Clear A                       | 1            | 1             |                  |                             |              |               |
| CPL A               | Complement A                  | 1            | 1             |                  |                             |              |               |
| DA A                | Decimal adjust A              | 1            | 1             |                  |                             |              |               |
| SWAP A              | Swap nibbles of A             | 1            | 1             |                  |                             |              |               |
| RL A                | Rotate A left                 | 1            | 1             |                  |                             |              |               |
| RLC A               | Rotate A left through carry   | 1            | 1             |                  |                             |              |               |
| RR A                | Rotate A right                | 1            | 1             |                  |                             |              |               |
| RRC A               | Rotate A right through carry  | 1            | 1             |                  |                             |              |               |
| <b>Input/Output</b> |                               |              |               | <b>Branch</b>    |                             |              |               |
| <b>Mnemonic</b>     | <b>Description</b>            | <b>Bytes</b> | <b>Cycles</b> | <b>Mnemonic</b>  | <b>Description</b>          | <b>Bytes</b> | <b>Cycles</b> |
| IN A, P             | Input port to A               | 1            | 2             | JMP addr         | Jump unconditional          | 2            | 2             |
| OUTL P, A           | Output A to port              | 1            | 2             | JMPP @A          | Jump indirect               | 1            | 2             |
| ANL P, # data       | And immediate to port         | 2            | 2             | DJNZ R, addr     | Decrement register and skip | 2            | 2             |
| ORL P, # data       | Or immediate to port          | 2            | 2             | JC addr          | Jump on carry = 1           | 2            | 2             |
| INS A, BUS          | Input BUS to A                | 1            | 2             | JNC addr         | Jump on carry = 0           | 2            | 2             |
| OUTL BUS, A         | Output A to BUS               | 1            | 2             | JZ addr          | Jump on A zero              | 2            | 2             |
| ANL BUS, # data     | And immediate to BUS          | 2            | 2             | JNZ addr         | Jump on A not zero          | 2            | 2             |
| ORL BUS, # data     | Or immediate to BUS           | 2            | 2             | JT0 addr         | Jump on T0 = 1              | 2            | 2             |
| MOVD A, P           | Input expander port to A      | 1            | 2             | JNT0 addr        | Jump on T0 = 0              | 2            | 2             |
| MOVD P, A           | Output A to expander port     | 1            | 2             | JT1 addr         | Jump on T1 = 1              | 2            | 2             |
| ANLD P, A           | And A to expander port        | 1            | 2             | JNT1 addr        | Jump on T1 = 0              | 2            | 2             |
| ORLD P, A           | Or A to expander port         | 1            | 2             | JF0 addr         | Jump on F0 = 1              | 2            | 2             |
|                     |                               |              |               | JF1 addr         | Jump on F1 = 1              | 2            | 2             |
|                     |                               |              |               | JTF addr         | Jump on timer flag          | 2            | 2             |
|                     |                               |              |               | JNI addr         | Jump on INT = 0             | 2            | 2             |
|                     |                               |              |               | JBb addr         | Jump on accumulator bit     | 2            | 2             |
| <b>Subroutine</b>   |                               |              |               | <b>Flags</b>     |                             |              |               |
| <b>Mnemonic</b>     | <b>Description</b>            | <b>Bytes</b> | <b>Cycles</b> | <b>Mnemonic</b>  | <b>Description</b>          | <b>Bytes</b> | <b>Cycles</b> |
| CALL addr           | Jump to subroutine            | 2            | 2             | CLR C            | Clear carry                 | 1            | 1             |
| RET                 | Return                        | 1            | 2             | CPL C            | Complement carry            | 1            | 1             |
| RETR                | Return and restore status     | 1            | 2             | CLR F0           | Clear flag 0                | 1            | 1             |
|                     |                               |              |               | CPL F0           | Complement flag 0           | 1            | 1             |
|                     |                               |              |               | CLR F1           | Clear flag 1                | 1            | 1             |
|                     |                               |              |               | CPL F1           | Complement flag 1           | 1            | 1             |

Table 2. Instruction Set (Continued)

| Data Moves     |                                   |       |        | Timer/Counter |                                 |       |        |
|----------------|-----------------------------------|-------|--------|---------------|---------------------------------|-------|--------|
| Mnemonic       | Description                       | Bytes | Cycles | Mnemonic      | Description                     | Bytes | Cycles |
| MOV A, R       | Move register to A                | 1     | 1      | MOV A, T      | Read timer/counter              | 1     | 1      |
| MOV A, @R      | Move data memory to A             | 1     | 1      | MOV T, A      | Load timer/counter              | 1     | 1      |
| MOV A, # data  | Move immediate to A               | 2     | 2      | STRT T        | Start timer                     | 1     | 1      |
| MOV R, A       | Move A to register                | 1     | 1      | STRT CNT      | Start counter                   | 1     | 1      |
| MOV @R, A      | Move A to data memory             | 1     | 1      | STOP TCNT     | Stop timer/counter              | 1     | 1      |
| MOV R, # data  | Move immediate to register        | 2     | 2      | EN TCNTI      | Enable timer/counter interrupt  | 1     | 1      |
| MOV @R, # data | Move immediate to data memory     | 2     | 2      | DIS TCNTI     | Disable timer/counter interrupt | 1     | 1      |
| MOV A, PSW     | Move PSW to A                     | 1     | 1      | Control       |                                 |       |        |
| MOV PSW, A     | Move A to PSW                     | 1     | 1      | Mnemonic      | Description                     | Bytes | Cycles |
| XCH A, R       | Exchange A and register           | 1     | 1      | EN I          | Enable external interrupt       | 1     | 1      |
| XCH A, @R      | Exchange A and data memory        | 1     | 1      | DIS I         | Disable external interrupt      | 1     | 1      |
| XCHD A, @R     | Exchange nibble of A and register | 1     | 1      | SEL RB0       | Select register bank 0          | 1     | 1      |
| MOVX A, @R     | Move external data memory to A    | 1     | 2      | SEL RB1       | Select register bank 1          | 1     | 1      |
| MOVX @R, A     | Move A to external data memory    | 1     | 2      | SEL MB0       | Select memory bank 0            | 1     | 1      |
| MOVP A, @A     | Move to A from current page       | 1     | 2      | SEL MB1       | Select memory bank 1            | 1     | 1      |
| MOVP3 A, @A    | Move to A from page 3             | 1     | 2      | ENT0 CLK      | Enable clock output on T0       | 1     | 1      |
| NOP            |                                   |       |        | Mnemonic      | Description                     | Bytes | Cycles |
|                |                                   |       |        | NOP           | No operation                    | 1     | 1      |

Ambient Temperature Under Bias .... 0°C to 70°C  
 Storage Temperature ..... -65°C to +150°C  
 Voltage On Any Pin With Respect  
 to Ground ..... -0.5V to +7V  
 Power Dissipation ..... 1.0 Watt

lute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of device at these or any other conditions above those indicated in the operational sections of this specification is not implied.

**D.C. CHARACTERISTICS:** ( $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ;  $V_{CC} = V_{DD} = 5V \pm 10\%$ ;  $V_{SS} = 0V$ )

| Symbol    | Parameter   | Limits |     |          | Unit | Test Conditions                    | Device |
|-----------|---|--------|-----|----------|------|------------------------------------|--------|
|           |   | Min    | Typ | Max      |      |                                    |        |
| $V_{IL}$  | Input Low Voltage (All Except RESET, X1, X2)        | - .5   |     | .8       | V    |                                    | All    |
| $V_{IL1}$ | Input Low Voltage (RESET, X1, X2)                   | - .5   |     | .6       | V    |                                    | All    |
| $V_{IH}$  | Input High Voltage (All Except XTAL1, XTAL2, RESET) | 2.0    |     | $V_{CC}$ | V    |                                    | All    |
| $V_{IH1}$ | Input High Voltage (X1, X2, RESET)                  | 3.8    |     | $V_{CC}$ | V    |                                    | All    |
| $V_{OL}$  | Output Low Voltage (BUS)                            |        |     | .45      | V    | $I_{OL} = 2.0\text{ mA}$           | All    |
| $V_{OL1}$ | Output Low Voltage (RD, WR, PSEN, ALE)              |        |     | .45      | V    | $I_{OL} = 1.8\text{ mA}$           | All    |
| $V_{OL2}$ | Output Low Voltage (PROG)                           |        |     | .45      | V    | $I_{OL} = 1.0\text{ mA}$           | All    |
| $V_{OL3}$ | Output Low Voltage (All Other Outputs)              |        |     | .45      | V    | $I_{OL} = 1.6\text{ mA}$           | All    |
| $V_{OH}$  | Output High Voltage (BUS)                           | 2.4    |     |          | V    | $I_{OH} = -400\text{ }\mu\text{A}$ | All    |
| $V_{OH1}$ | Output High Voltage (RD, WR, PSEN, ALE)             | 2.4    |     |          | V    | $I_{OH} = -100\text{ }\mu\text{A}$ | All    |
| $V_{OH2}$ | Output High Voltage (All Other Outputs)             | 2.4    |     |          | V    | $I_{OH} = -40\text{ }\mu\text{A}$  | All    |

**D.C. CHARACTERISTICS:** ( $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ;  $V_{CC} = V_{DD} = 5\text{V} \pm 10\%$ ;  $V_{SS} = 0\text{V}$ ) (Continued)

| Symbol            | Parameter  | Limits |     |          | Unit          | Test Conditions                        | Device |
|-------------------|--|--------|-----|----------|---------------|--|--------|
|                   |  | Min    | Typ | Max      |               |  |        |
| $I_{L1}$          | Leakage Current<br>(T1, INT)                           |        |     | $\pm 10$ | $\mu\text{A}$ | $V_{SS} \leq V_{IN} \leq V_{CC}$       | All    |
| $I_{LI1}$         | Input Leakage Current<br>(P10–P17, P20–P27,<br>EA, SS) |        |     | –500     | $\mu\text{A}$ | $V_{SS} + .45 \leq V_{IN} \leq V_{CC}$ | All    |
| $I_{LI2}$         | Input Leakage Current<br>RESET                         | –10    |     | –300     | $\mu\text{A}$ | $V_{SS} \leq V_{IN} \leq 3.8\text{V}$  | All    |
| $I_{L0}$          | Leakage Current<br>(BUS, T0) (High<br>Impedance State) |        |     | $\pm 10$ | $\mu\text{A}$ | $V_{SS} \leq V_{IN} \leq V_{CC}$       | All    |
| $I_{DD} + I_{CC}$ | Total Supply Current*                                  |        | 80  | 100      | mA            |  | 8035H  |
|                   |  |        | 95  | 110      | mA            |  | 8039H  |
|                   |  |        | 80  | 100      | mA            |  | 8748H  |
|                   |  |        | 95  | 110      | mA            |  | 8749H  |

\* $I_{CC} + I_{DD}$  is measured with all outputs disconnected; SS, RESET, and INT equal to  $V_{CC}$ ; EA equal to  $V_{SS}$ .



**A.C. CHARACTERISTICS:** ( $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ;  $V_{CC} = V_{DD} = 5V \pm 10\%$ ;  $V_{SS} = 0V$ )

| Symbol             | Parameter  | f (t)<br>(Note 3) | 11 MHz |      | Unit          | Conditions<br>(Note 1) |
|--------------------|--|-------------------|--------|------|---------------|------------------------|
|                    |  |                   | Min    | Max  |               |                        |
| t                  | Clock Period   | 1/xtal freq       | 90.9   | 1000 | ns            | (Note 3)               |
| t <sub>LL</sub>    | ALE Pulse Width  | 3.5t-170          | 150    |      | ns            |                        |
| t <sub>AL</sub>    | Addr Setup to ALE  | 2t-110            | 70     |      | ns            | (Note 2)               |
| t <sub>LA</sub>    | Addr Hold from ALE   | t-40              | 50     |      | ns            |                        |
| t <sub>CC1</sub>   | Control Pulse Width ( $\overline{RD}$ , $\overline{WR}$ )                | 7.5t-200          | 480    |      | ns            |                        |
| t <sub>CC2</sub>   | Control Pulse Width ( $\overline{PSEN}$ )                                | 6t-200            | 350    |      | ns            |                        |
| t <sub>DW</sub>    | Data Setup before $\overline{WR}$  | 6.5t-200          | 390    |      | ns            |                        |
| t <sub>WD</sub>    | Data Hold after $\overline{WR}$  | t-50              | 40     |      | ns            |                        |
| t <sub>DR</sub>    | Data Hold ( $\overline{RD}$ , $\overline{PSEN}$ )                        | 1.5t-30           | 0      | 110  | ns            |                        |
| t <sub>RD1</sub>   | $\overline{RD}$ to Data in   | 6t-170            |        | 375  | ns            |                        |
| t <sub>RD2</sub>   | $\overline{PSEN}$ to Data in   | 4.5t-170          |        | 240  | ns            |                        |
| t <sub>AW</sub>    | Addr Setup to $\overline{WR}$  | 5t-150            | 300    |      | ns            |                        |
| t <sub>AD1</sub>   | Addr Setup to Data ( $\overline{RD}$ )                                   | 10.5t-220         |        | 730  | ns            |                        |
| t <sub>AD2</sub>   | Addr Setup to Data ( $\overline{PSEN}$ )                                 | 7.5t-200          |        | 460  | ns            |                        |
| t <sub>AFC1</sub>  | Addr Float to $\overline{RD}$ , $\overline{WR}$                          | 2t-40             | 140    |      | ns            | (Note 2)               |
| t <sub>AFC2</sub>  | Addr Float to $\overline{PSEN}$  | .5t-40            | 10     |      | ns            | (Note 2)               |
| t <sub>LAFC1</sub> | ALE to Control ( $\overline{RD}$ , $\overline{WR}$ )                     | 3t-75             | 200    |      | ns            |                        |
| t <sub>LAFC2</sub> | ALE to Control ( $\overline{PSEN}$ )                                     | 1.5t-75           | 60     |      | ns            |                        |
| t <sub>CA1</sub>   | Control to ALE ( $\overline{RD}$ , $\overline{WR}$ , $\overline{PROG}$ ) | t-65              | 25     |      | ns            |                        |
| t <sub>CA2</sub>   | Control to ALE ( $\overline{PSEN}$ )                                     | 4t-70             | 290    |      | ns            |                        |
| t <sub>CP</sub>    | Port Control Setup to $\overline{PROG}$                                  | 1.5t-80           | 50     |      | ns            |                        |
| t <sub>PC</sub>    | Port Control Hold to $\overline{PROG}$                                   | 4t-260            | 100    |      | ns            |                        |
| t <sub>PR</sub>    | $\overline{PROG}$ to P2 Input Valid                                      | 8.5t-120          |        | 650  | ns            |                        |
| t <sub>PF</sub>    | Input Data Hold from $\overline{PROG}$                                   | 1.5t              | 0      | 140  | ns            |                        |
| t <sub>DP</sub>    | Output Data Setup  | 6t-290            | 250    |      | ns            |                        |
| t <sub>PD</sub>    | Output Data Hold   | 1.5t-90           | 40     |      | ns            |                        |
| t <sub>PP</sub>    | $\overline{PROG}$ Pulse Width  | 10.5t-250         | 700    |      | ns            |                        |
| t <sub>PL</sub>    | Port 2 I/O Setup to ALE  | 4t-200            | 160    |      | ns            |                        |
| t <sub>LP</sub>    | Port 2 I/O Hold to ALE   | .5t-30            | 15     |      | ns            |                        |
| t <sub>PV</sub>    | Port Output from ALE   | 4.5t+100          |        | 510  | ns            |                        |
| t <sub>OPRR</sub>  | T0 Rep Rate  | 3t                | 270    |      | ns            |                        |
| t <sub>CY</sub>    | Cycle Time   | 15t               | 1.36   | 15.0 | $\mu\text{s}$ |                        |

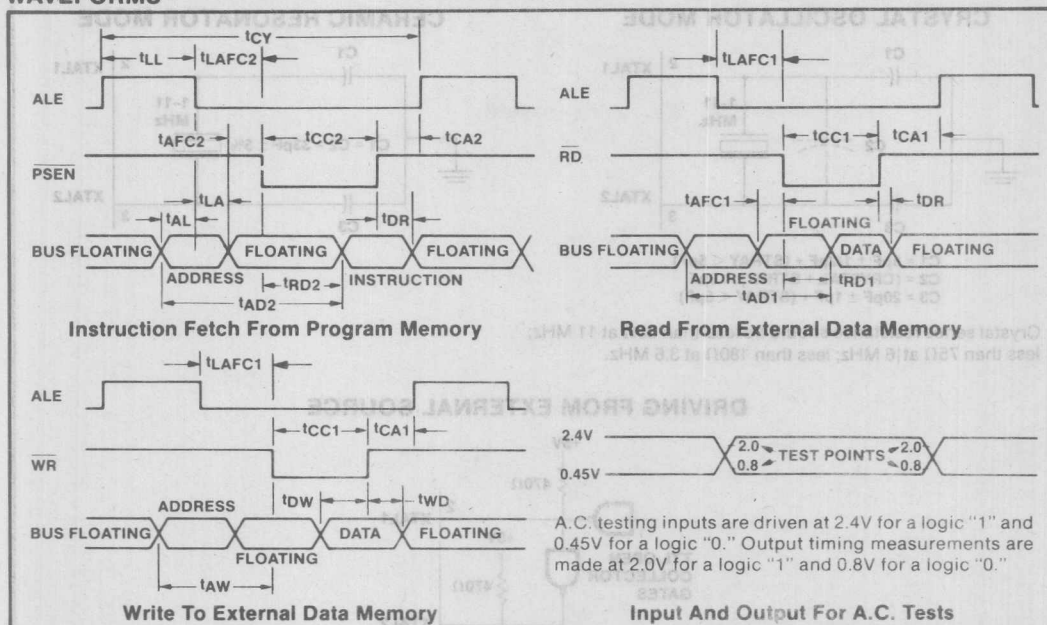
**Notes:**

1. Control Outputs  $CL = 80\text{pF}$   
BUS Outputs  $CL = 150\text{pF}$

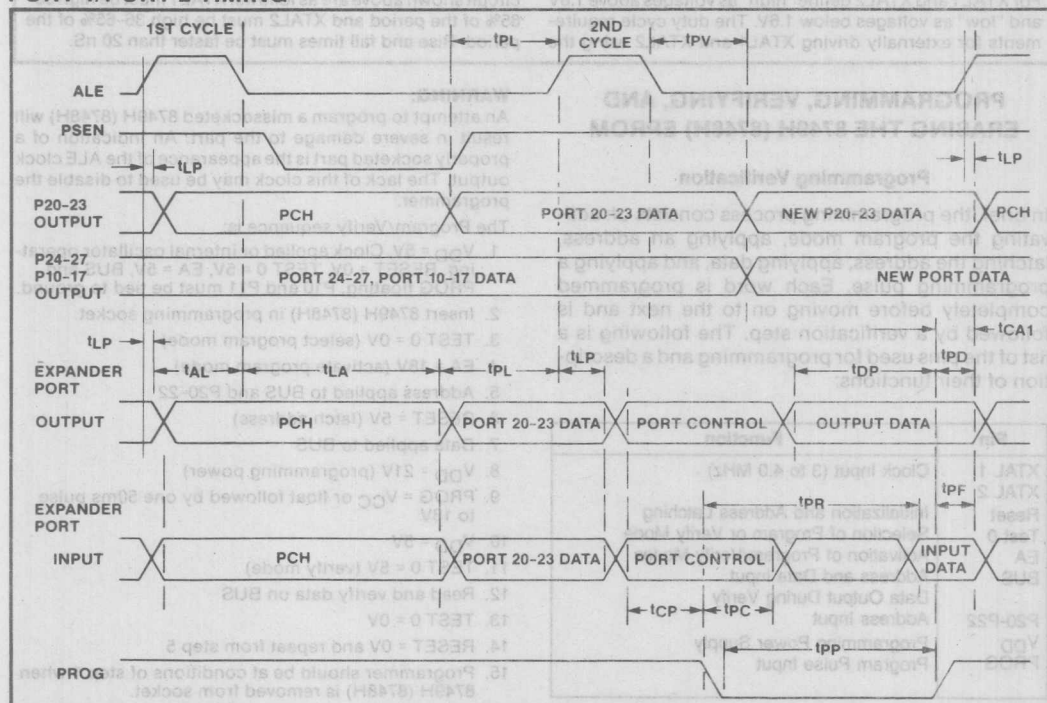
2. BUS High Impedance  
Load  $20\text{pF}$

3. f(t) assumes 50% duty cycle on X1, X2. Max  
clock period is for a 1 MHz crystal input.

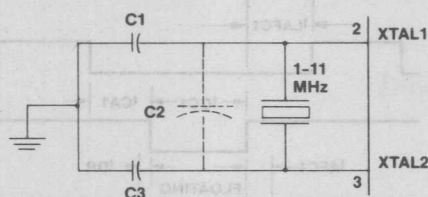
# WAVEFORMS



# PORT 1/PORT 2 TIMING



### CRYSTAL OSCILLATOR MODE



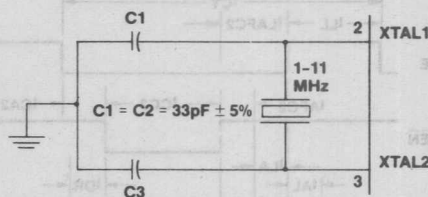
$$C1 = 5\text{pF} + 1/2\text{pF} + (\text{STRAY} < 5\text{pF})$$

$$C2 = (\text{CRYSTAL} + \text{STRAY}) < 8\text{pF}$$

$$C3 = 20\text{pF} + 1\text{pF} + (\text{STRAY} < 5\text{pF})$$

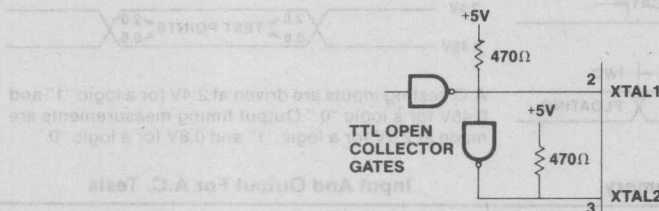
Crystal series resistance should be less than 30Ω at 11 MHz;  
less than 75Ω at 6 MHz; less than 180Ω at 3.6 MHz.

### CERAMIC RESONATOR MODE



$$C1 = C2 = 33\text{pF} \pm 5\%$$

### DRIVING FROM EXTERNAL SOURCE



For XTAL1 and XTAL2 define "high" as voltages above 1.6V and "low" as voltages below 1.6V. The duty cycle requirements for externally driving XTAL1 and XTAL2 using the

circuit shown above are as follows: XTAL1 must be high 35-65% of the period and XTAL2 must be high 36-65% of the period. Rise and fall times must be faster than 20 nS.

### PROGRAMMING, VERIFYING, AND ERASING THE 8749H (8748H) EPROM

#### Programming Verification

In brief, the programming process consists of: activating the program mode, applying an address, latching the address, applying data, and applying a programming pulse. Each word is programmed completely before moving on to the next and is followed by a verification step. The following is a list of the pins used for programming and a description of their functions:

| Pin              | Function                            |
|------------------|-------------------------------------|
| XTAL 1<br>XTAL 2 | Clock Input (3 to 4.0 MHz)          |
| Reset            | Initialization and Address Latching |
| Test 0           | Selection of Program or Verify Mode |
| EA               | Activation of Program/Verify Modes  |
| BUS              | Address and Data Input              |
|                  | Data Output During Verify           |
| P20-P22          | Address Input                       |
| VDD              | Programming Power Supply            |
| PROG             | Program Pulse Input                 |

#### WARNING:

An attempt to program a missocketed 8749H (8748H) will result in severe damage to the part. An indication of a properly socketed part is the appearance of the ALE clock output. The lack of this clock may be used to disable the programmer.

The Program/Verify sequence is:

1.  $V_{DD} = 5V$ , Clock applied or internal oscillator operating. RESET = 0V, TEST 0 = 5V, EA = 5V, BUS and PROG floating. P10 and P11 must be tied to ground.
2. Insert 8749H (8748H) in programming socket.
3. TEST 0 = 0V (select program mode)
4. EA = 18V (activate program mode)
5. Address applied to BUS and P20-22
6. RESET = 5V (latch address)
7. Data applied to BUS
8.  $V_{DD} = 21V$  (programming power)
9. PROG =  $V_{CC}$  or float followed by one 50ms pulse to 18V
10.  $V_{DD} = 5V$
11. TEST 0 = 5V (verify mode)
12. Read and verify data on BUS
13. TEST 0 = 0V
14. RESET = 0V and repeat from step 5
15. Programmer should be at conditions of step 1 when 8749H (8748H) is removed from socket.

**A.C. TIMING SPECIFICATION FOR PROGRAMMING 8748H/8749H ONLY:**

( $T_A = 25^\circ\text{C} \pm 5^\circ\text{C}$ ;  $V_{CC} = 5\text{V} \pm 5\%$ ;  $V_{DD} = 21 \pm .5\text{V}$ )

| Symbol     | Parameter  | Min       | Max       | Unit          | Test Conditions |
|------------|--|-----------|-----------|---------------|-----------------|
| $t_{AW}$   | Address Setup Time to $\overline{\text{RESET}}$                    | $4t_{CY}$ |           |               |                 |
| $t_{WA}$   | Address Hold Time After $\overline{\text{RESET}}$                  | $4t_{CY}$ |           |               |                 |
| $t_{DW}$   | Data in Setup Time to $\overline{\text{PROG}}$                     | $4t_{CY}$ |           |               |                 |
| $t_{WD}$   | Data in Hold Time After $\overline{\text{PROG}}$                   | $4t_{CY}$ |           |               |                 |
| $t_{PH}$   | $\overline{\text{RESET}}$ Hold Time to Verify                      | $4t_{CY}$ |           |               |                 |
| $t_{VDDW}$ | $V_{DD}$ Hold Time Before $\overline{\text{PROG}}$                 | 0         | 1.0       | ms            |                 |
| $t_{VDDH}$ | $V_{DD}$ Hold Time After $\overline{\text{PROG}}$                  | 0         | 1.0       | ms            |                 |
| $t_{PW}$   | Program Pulse Width  | 50        | 60        | ms            |                 |
| $t_{TW}$   | Test 0 Setup Time for Program Mode                                 | $4t_{CY}$ |           |               |                 |
| $t_{WT}$   | Test 0 Hold Time After Program Mode                                | $4t_{CY}$ |           |               |                 |
| $t_{DO}$   | Test 0 to Data Out Delay   |           | $4t_{CY}$ |               |                 |
| $t_{WW}$   | $\overline{\text{RESET}}$ Pulse Width to Latch Address             | $4t_{CY}$ |           |               |                 |
| $t_r, t_f$ | $V_{DD}$ and $\overline{\text{PROG}}$ Rise and Fall Times          | 0.5       | 100       | $\mu\text{s}$ |                 |
| $t_{CY}$   | CPU Operation Cycle Time   | 3.75      | 5         | $\mu\text{s}$ |                 |
| $t_{RE}$   | $\overline{\text{RESET}}$ Setup Time before $\overline{\text{EA}}$ | $4t_{CY}$ |           |               |                 |

**NOTE:** If Test 0 is high,  $t_{DO}$  can be triggered by  $\overline{\text{RESET}}$ .

**D.C. TIMING SPECIFICATION FOR PROGRAMMING 8748H/8749H ONLY:**

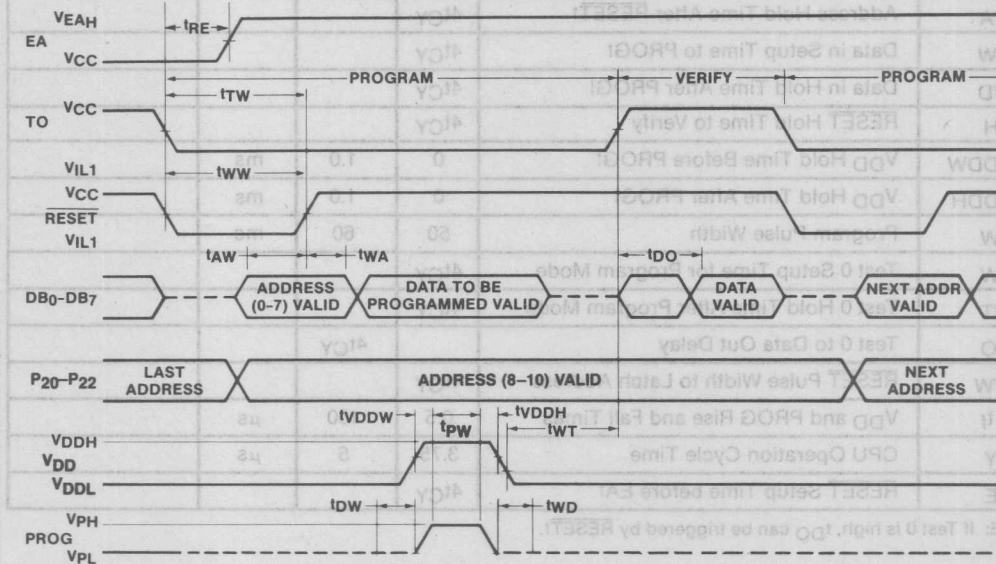
( $T_A = 25^\circ\text{C} \pm 5^\circ\text{C}$ ;  $V_{CC} = 5\text{V} \pm 5\%$ ;  $V_{DD} = 21 \pm .5\text{V}$ )

| Symbol     | Parameter   | Min  | Max      | Unit | Test Conditions |
|------------|---|------|----------|------|-----------------|
| $V_{DDH}$  | $V_{DD}$ Program Voltage High Level                         | 20.5 | 21.5     | V    |                 |
| $V_{DDL}$  | $V_{DD}$ Voltage Low Level                                  | 4.75 | 5.25     | V    |                 |
| $V_{PH}$   | $\overline{\text{PROG}}$ Program Voltage High Level         | 17.5 | 18.5     | V    |                 |
| $V_{PL}$   | $\overline{\text{PROG}}$ Voltage Low Level                  | 4.0  | $V_{CC}$ | V    |                 |
| $V_{EAH}$  | $\overline{\text{EA}}$ Program or Verify Voltage High Level | 17.5 | 18.5     | V    |                 |
| $I_{DD}$   | $V_{DD}$ High Voltage Supply Current                        |      | 20.0     | mA   |                 |
| $I_{PROG}$ | $\overline{\text{PROG}}$ High Voltage Supply Current        |      | 1.0      | mA   |                 |
| $I_{EA}$   | $\overline{\text{EA}}$ High Voltage Supply Current          |      | 1.0      | mA   |                 |

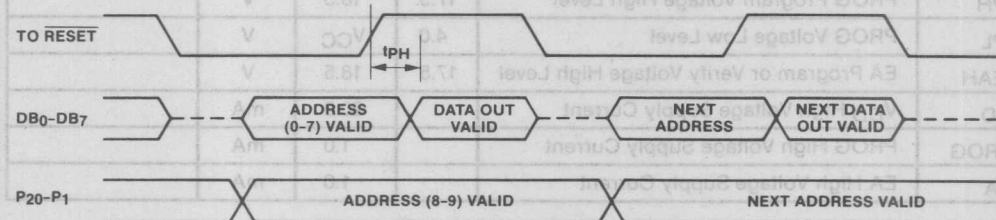


# WAVEFORMS

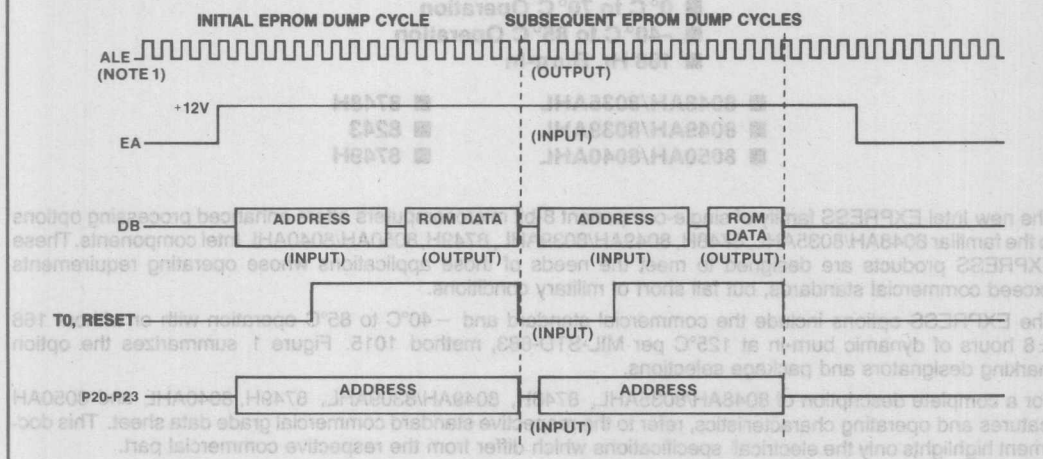
## COMBINATION PROGRAM/VERIFY MODE (EPROM'S ONLY)



## VERIFY MODE



### SUGGESTED EPROM VERIFICATION ALGORITHM FOR HMOS-E DEVICE ONLY



|  |  |  | Temp Range<br>C |  |  | 0 Hrs            |  |  | 0 Yr  |  |  | -40 to -85 |  |  |
|--|--|--|-----------------|--|--|------------------|--|--|-------|--|--|------------|--|--|
|  |  |  | 0 Hrs           |  |  | 0 Hrs            |  |  | 0 Hrs |  |  | 0 Hrs      |  |  |
|  |  |  | VSS = 0V        |  |  | VCC = VDD = + 5V |  |  |       |  |  |            |  |  |
|  |  |  | ADDR            |  |  | 49H              |  |  | 48H   |  |  |            |  |  |
|  |  |  | A10             |  |  | 0                |  |  | 0     |  |  |            |  |  |
|  |  |  | A11             |  |  | 0                |  |  | 0     |  |  |            |  |  |

**NOTE:** ALE is function of X1, X2 inputs.

**NOTE:** ALE is function of X1, X2 inputs.

# SINGLE-COMPONENT 8-BIT MICROCOMPUTERS EXPRESS

- 0°C to 70°C Operation
- -40°C to 85°C Operation
- 168 Hr. Burn-In

- 8048AH/8035AHL
- 8049AH/8039AHL
- 8050AH/8040AHL
- 8748H
- 8243
- 8749H

The new Intel EXPRESS family of single-component 8-bit microcomputers offers enhanced processing options to the familiar 8048AH/8035AHL, 8748H, 8049AH/8039AHL, 8749H, 8050AH/8040AHL Intel components. These EXPRESS products are designed to meet the needs of those applications whose operating requirements exceed commercial standards, but fall short of military conditions.

The EXPRESS options include the commercial standard and -40°C to 85°C operation with or without 168 ± 8 hours of dynamic burn-in at 125°C per MIL-STD-883, method 1015. Figure 1 summarizes the option marking designators and package selections.

For a complete description of 8048AH/8035AHL, 8748H, 8049AH/8039AHL, 8749H, 8040AHL and 8050AH features and operating characteristics, refer to the respective standard commercial grade data sheet. This document highlights only the electrical specifications which differ from the respective commercial part.

| Temp Range<br>C° | 0-70     | -40-85    | 0-70      | -40-85    |
|------------------|----------|-----------|-----------|-----------|
|                  | 0 Hrs    | 0 Hrs     | 168 Hrs   | 168 Hrs   |
| Burn In          |          |           |           |           |
|                  | P8048AH  | TP8048AH  | QP8048AH  | LP8048AH  |
|                  | D8048AH  | TD8048AH  | QD8048AH  | LD8048AH  |
|                  | D8748H   | TD8748H   | QD8748H   | LD8748H   |
|                  | P8035AHL | TP8035AHL | QP8035AHL | LP8035AHL |
|                  | D8035AHL | TD8035AHL | QD8035AHL | LD8035AHL |
|                  | P8049AH  | TP8049AH  | QP8049AH  | LP8049AH  |
|                  | D8049AH  | TD8049AH  | QD8049AH  | LD8049AH  |
|                  | D8749H   | TD8749H   | QD8749H   | LD8749H   |
|                  | P8039AHL | TP8039AHL | QP8039AHL | LP8039AHL |
|                  | D8039AHL | TD8039AHL | QD8039AHL | LD8039AHL |
|                  | P8050AH  | TP8050AH  | QP8050AH  | LP8050AH  |
|                  | D8050AH  | TD8050AH  | QD8050AH  | LD8050AH  |
|                  | P8040AHL | TP8040AHL | QP8040AHL | LP8040AHL |
|                  | D8040AHL | TD8040AHL | QD8040AHL | LD8040AHL |
|                  | P8243    | TP8243    | QP8243    | —         |
|                  | D8243    | TD8243    | QD8243    | LD8243    |

\* Commercial Grade  
P Plastic Package  
D Cerdip Package

Extended Temperature Electrical Specification Deviations\*

TP8048AH/TP8035AHL/LP8048AH/LP8035AHL  
TD8048AH/TD8035AHL/LD8048AH/LD8035AHL

**D.C. CHARACTERISTICS:** ( $T_A = -40^\circ\text{C}$  to  $85^\circ\text{C}$ ;  $V_{CC} = V_{DD} = 5\text{V} \pm 10\%$ ;  $V_{SS} = 0\text{V}$ )

| Symbol            | Parameter   | Limits |     |          | Unit | Test Conditions |
|-------------------|---|--------|-----|----------|------|-----------------|
|                   |   | Min    | Typ | Max      |      |                 |
| $V_{IH}$          | Input High Voltage (All Except XTAL1, XTAL2, RESET) | 2.2    |     | $V_{CC}$ | V    |                 |
| $I_{DD}$          | $V_{DD}$ Supply Current                             |        | 4   | 8        | mA   |                 |
| $I_{DD} + I_{CC}$ | Total Supply Current                                |        | 40  | 80       | mA   |                 |

TP8049AH/TP8039AHL/LP8049AH/LP8039AHL  
TD8049AH/TD8039AHL/LD8049AH/LD8039AHL

**D.C. CHARACTERISTICS:** ( $T_A = -40^\circ\text{C}$  to  $85^\circ\text{C}$ ;  $V_{CC} = V_{DD} = 5\text{V} \pm 10\%$ ;  $V_{SS} = 0\text{V}$ )

| Symbol            | Parameter   | Limits |     |          | Unit | Test Conditions |
|-------------------|---|--------|-----|----------|------|-----------------|
|                   |   | Min    | Typ | Max      |      |                 |
| $V_{IH}$          | Input High Voltage (All Except XTAL1, XTAL2, RESET) | 2.2    |     | $V_{CC}$ | V    |                 |
| $I_{DD}$          | $V_{DD}$ Supply Current                             |        | 5   | 10       | mA   |                 |
| $I_{DD} + I_{CC}$ | Total Supply Current                                |        | 50  | 100      | mA   |                 |

TP8050AH/TP8040AHL/LP8050AHL/LP8040AHL  
TD8050AH/TD8040AHL/LD8050AHL/LD8040AHL

**D.C. CHARACTERISTICS:** ( $T_A = -40^\circ\text{C}$  to  $85^\circ\text{C}$ ;  $V_{CC} = V_{DD} = 5\text{V} \pm 10\%$ ;  $V_{SS} = 0\text{V}$ )

| Symbol            | Parameter   | Limits |     |          | Unit | Test Conditions |
|-------------------|---|--------|-----|----------|------|-----------------|
|                   |   | Min    | Typ | Max      |      |                 |
| $V_{IH}$          | Input High Voltage (All Except XTAL1, XTAL2, RESET) | 2.2    |     | $V_{CC}$ | V    |                 |
| $I_{DD}$          | $V_{DD}$ Supply Current                             |        | 10  | 20       | mA   |                 |
| $I_{DD} + I_{CC}$ | Total Supply Current                                |        | 75  | 120      | mA   |                 |



## Extended Temperature Electrical Specification Deviations\*

## TD8748H/LD8748H

D.C. CHARACTERISTICS: ( $T_A = -40^\circ\text{C}$  to  $85^\circ\text{C}$ ;  $V_{CC} = V_{DD} = 5V \pm 10\%$ ;  $V_{SS} = 0V$ )

| Symbol            | Parameter   | Limits |     |          | Unit | Test Conditions |
|-------------------|---|--------|-----|----------|------|-----------------|
|                   |   | Min    | Typ | Max      |      |                 |
| $V_{IH}$          | Input High Voltage (All Except XTAL1, XTAL2, RESET) | 2.2    |     | $V_{CC}$ | V    |                 |
| $I_{DD} + I_{CC}$ | Total Supply Current                                |        | 50  | 130      | mA   |                 |

## TD8749H/LD8749H

D.C. CHARACTERISTICS: ( $T_A = -40^\circ\text{C}$  to  $85^\circ\text{C}$ ;  $V_{CC} = V_{DD} = 5V \pm 10\%$ ;  $V_{SS} = 0V$ )

| Symbol            | Parameter   | Limits |     |          | Unit | Test Conditions |
|-------------------|---|--------|-----|----------|------|-----------------|
|                   |   | Min    | Typ | Max      |      |                 |
| $V_{IH}$          | Input High Voltage (All Except XTAL1, XTAL2, RESET) | 2.2    |     | $V_{CC}$ | V    |                 |
| $I_{DD} + I_{CC}$ | Total Supply Current                                |        | 75  | 150      | mA   |                 |

## TP8243/TD8243/LD8243

D.C. CHARACTERISTICS: ( $T_A = -40^\circ\text{C}$  to  $85^\circ\text{C}$ ;  $V_{CC} = 5V \pm 10\%$ ;  $V_{SS} = 0V$ )

| Symbol            | Parameter   | Limits |     |          | Unit | Test Conditions |
|-------------------|---|--------|-----|----------|------|-----------------|
|                   |   | Min    | Typ | Max      |      |                 |
| $I_{CC}$          | $V_{CC}$ Supply Current                             |        | 15  | 25       | mA   |                 |
| $V_{IH}$          | Input High Voltage (All Except XTAL1, XTAL2, RESET) | 2.2    |     | $V_{CC}$ | V    |                 |
| $I_{DD}$          | $V_{DD}$ Supply Current                             |        | 10  | 20       | mA   |                 |
| $I_{DD} + I_{CC}$ | Total Supply Current                                |        | 25  | 50       | mA   |                 |

## Extended Temperature Electrical Specification Deviations\*

TD8022/LD8022

**D.C. CHARACTERISTICS:** ( $T_A = -40^\circ\text{C}$  to  $85^\circ\text{C}$ ;  $V_{CC} = 5.5\text{V} \pm 1\text{V}$ ;  $V_{SS} = 0\text{V}$ )

| Symbol    | Parameter  | Limits       |     |              | Unit          | Test Conditions   |
|-----------|--|--------------|-----|--------------|---------------|---|
|           |  | Min          | Typ | Max          |               |   |
| $V_{IL1}$ | Input Low Voltage (Port 0)                                   | -0.5         |     | $V_{TH}-0.2$ | V             |   |
| $V_{IH}$  | Input High Voltage<br>(All Except XTAL1, RESET)              | 2.3          |     | $V_{CC}$     | V             | $V_{CC} = 5.0\text{V} \pm 10\%$<br>$V_{TH}$ Floating      |
| $V_{IH1}$ | Input High Voltage<br>(All Except XTAL1, RESET)              | 3.8          |     | $V_{CC}$     | V             | $V_{CC} = 5.5\text{V} \pm 1\text{V}$<br>$V_{TH}$ Floating |
| $V_{IH2}$ | Input High Voltage (Port 0)                                  | $V_{TH}+0.2$ |     | $V_{CC}$     | V             |   |
| $V_{IH3}$ | Input High Voltage (RESET, XTAL1)                            | 3.8          |     | $V_{CC}$     | V             |   |
| $V_{IL}$  | Input Low Voltage  | -0.5         |     | 0.5          | V             |   |
| $V_{OL}$  | Output Low Voltage   |              |     | 0.45         | V             | $I_{OL} = 0.8\text{ mA}$                                  |
| $V_{OL1}$ | Output Low Voltage (P10, P11)                                |              |     | 2.5          | V             | $I_{OL} = 3\text{ mA}$                                    |
| $V_{OH}$  | Output High Voltage (All unless<br>open drain option Port 0) | 2.4          |     |              | V             | $I_{OH} = 30\text{ }\mu\text{A}$                          |
| $I_{LI}$  | Input Current (T1)   |              |     | $\pm 700$    | $\mu\text{A}$ | $V_{CC} \geq V_{IN} \geq$<br>$V_{SS} + 0.45\text{V}$      |
| $I_{LI1}$ | Input Current to Ports                                       |              |     | 500          | $\mu\text{A}$ | $V_{IN} = 0.45\text{V}$                                   |
| $I_{CC}$  | $V_{CC}$ Supply Current                                      |              |     | 120          | mA            |   |

**A.C. CHARACTERISTICS:** ( $T_A = -40^\circ\text{C}$  to  $85^\circ\text{C}$ ;  $V_{CC} = 5.5\text{V} \pm 1\text{V}$ ;  $V_{SS} = 0\text{V}$ )

| Symbol   | Parameter                                    | Min  | Max       | Unit          | Test Conditions                                   |
|----------|--|------|-----------|---------------|---|
| $t_{CY}$ | Cycle Time                                   | 8.38 | 50.0      | $\mu\text{s}$ | 3.58 MHz XTAL = $8.38\text{ }\mu\text{s } t_{CY}$ |
| $V_{T1}$ | Zero-Cross Detection Input (T1)              | 1    | 3         | VACpp         | AC Coupled  |
| AZC      | Zero-Cross Accuracy                          |      | $\pm 200$ | mV            | 60 Hz Sine Wave                                   |
| $F_{T1}$ | Zero-Cross Detection Input<br>Frequency (T1) | 0.05 | 1         | kHz           |   |
| $t_{LL}$ | ALE Pulse Width                              | 3.9  | 23.0      | $\mu\text{s}$ | $t_{CY} = 8.38\text{ }\mu\text{s}$ for min        |

**NOTE:** Control Outputs:  $C_L = 80\text{ pf}$ ;  $T_{CY} = 8.38\text{ }\mu\text{sec}$ .**A/D CONVERTER CHARACTERISTICS:** ( $AV_{CC} = 5.5\text{V} \pm 1\text{V}$ ;  $AV_{SS} = 0\text{V}$ ;  $AV_{CC}/2 \leq V_{AREF} \leq AV_{CC}$ )

| Parameter         | Limits |     |   | Unit | Test Conditions |
|-------------------|--------|-----|---|------|-----------------|
|                   | Min    | Typ | Max   |      |                 |
| Absolute Accuracy |        |     | $1.6\% \text{ FSR} \pm \frac{1}{2} \text{ LSB}$ | LSB  |                 |

**NOTE:** The analog input must be maintained at a constant voltage during the sample time ( $t_{ss} + t_{sh}$ ).

\*Refer to individual commercial grade data sheets for complete operating characteristics.

# 80C49-7/80C39-7

## CHMOS SINGLE-COMPONENT 8-BIT MICROCOMPUTER

- 80C49-7 Low Power Mask Programmable ROM
- 80C39-7 Low Power, CPU only

- Pin-to-pin Compatible with Intel's 8049AH/8039AHL
- 1.36  $\mu$ sec Instruction Cycle. All Instructions 1 or 2 Cycles
- Ability to Maintain Operation during AC Power Line Interruptions
- Exit Idle Mode with an External or Internal Interrupt Signal
- Battery Operation
- 3 Power Consumption Selections
  - Normal Operation: 12 mA @ 11 MHz @ 5V
  - Idle Mode: 5 mA @ 11 MHz @ 5V
  - Power Down: 2  $\mu$ A @ 2.0V
- 11 MHz, TTL Compatible Operation:
  - $V_{CC} = 5V \pm 10\%$
  - CMOS Compatible Operation;
    - $V_{CC} = 5V \pm 20\%$

Intel's 80C49-7/80C39-7 are low power, CHMOS versions of the popular MCS<sup>®</sup>-48 HMOS family members. CHMOS is a technology built on HMOS II and features high resistivity P substrate, diffused N well, and scaled N and P channel devices. The 80C49-7/80C39-7 have been designed to provide low power consumption and high performance.

The 80C49-7 contains a 2K x 8 program memory, a 128 x 8 x 8 RAM data memory, 27 I/O lines, and an 8-bit timer/counter in addition to an on-board oscillator and clock circuits. For systems that require extra capability, the 80C49-7 can be expanded using CMOS external memories and MCS<sup>®</sup>-80 and MCS<sup>®</sup>-85 peripherals. The 80C39-7 is the equivalent of the 80C49-7 without program memory on-board.

The CHMOS design of the 80C49-7 opens new application areas that require battery operation, low power standby, wide voltage range, and the ability to maintain operation during AC power line interruptions. These applications include portable and hand-held instruments, telecommunications, consumer, and automotive.

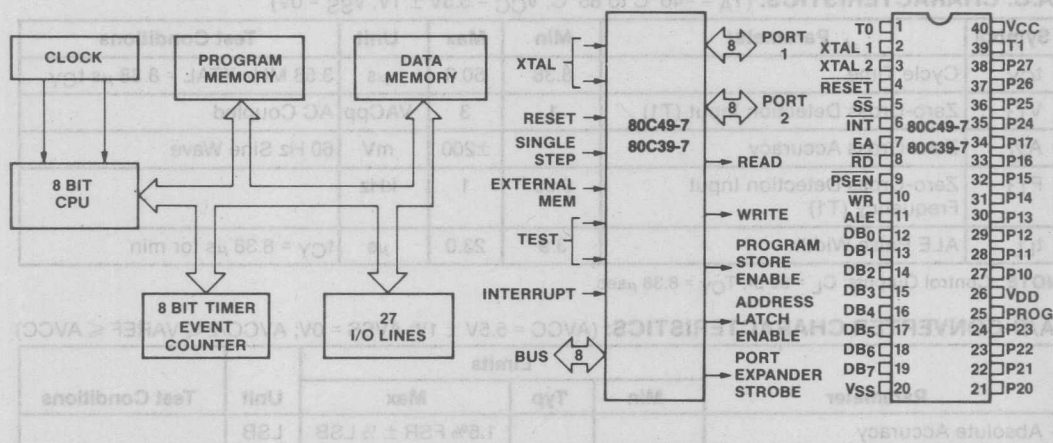


Figure 1.  
Block Diagram

Figure 2.  
Logic Symbol

Figure 3.  
Pin Configuration

Table 1. Pin Description

| Symbol         | Pin No. | Function  |
|----------------|---------|---|
| VSS            | 20      | Circuit GND potential   |
| VDD            | 26      | Low Power standby pin   |
| VCC            | 40      | Main power supply; +5V during operation.  |
| PROG           | 25      | Output strobe for 82C43 I/O expander.   |
| P10-P17 Port 1 | 27-34   | 8-bit quasi-bidirectional port.   |
| P20-P23        | 21-24   | 8-bit quasi-bidirectional port.   |
| P24-P27 Port 2 | 35-38   | P20-P23 contain the four high order program counter bits during an external program memory fetch and serve as a 4-bit I/O expander bus for 8243.  |
| DB0-DB7 BUS    | 12-19   | True bidirectional port which can be written or read synchronously using the RD, WR strobes. The port can also be statically latched.<br><br>Contains the 8 low order program counter bits during an external program memory fetch, and receives the addressed instruction under the control of PSEN. Also contains the address and data during an external RAM data store instruction, under control of ALE, RD, and WR. |
| T0             | 1       | Input pin testable using the conditional transfer instructions JT0 and JNT0. T0 can be designated as a clock output using ENT0 CLK instruction.   |
| T1             | 39      | Input pin testable using the JT1, and JNT1 instructions.  |

| Symbol | Pin No. | Function   |
|--------|---------|--|
|        |         | Can be designated the timer/counter input using the STRT CNT instruction.  |
| INT    | 6       | Interrupt input. Initiates an interrupt if interrupt is enabled. Interrupt is disabled after a reset. Also testable with conditional jump instruction. (Active low)<br>Interrupt must remain low for at least 3 machine cycles for proper operation. |
| RD     | 8       | Output strobe activated during a BUS read. Can be used to enable data onto the bus from an external device.<br><br>Used as a read strobe to external data memory. (Active low)   |
| RESET  | 4       | Input which is used to initialize the processor. (Active low) (Non TTL V <sub>IH</sub> )   |
| WR     | 10      | Output strobe during a bus write. (Active low)<br><br>Used as write strobe to external data memory.  |
| ALE    | 11      | Address latch enable. This signal occurs once during each cycle and is useful as a clock output.<br><br>The negative edge of ALE strobes address into external data and program memory.  |
| PSEN   | 9       | Program store enable. This output occurs only during a fetch to external program memory. (Active low)  |
| SS     | 5       | Single step input can be used in conjunction with  |



Table 1. Pin Description (Continued)

| Symbol     | Pin No. | Function  |
|------------|---------|---|
| SS (Con't) |         | ALE to "single step" the processor through each instruction (Active low)  |
| EA         | 7       | External access input which forces all program memory fetches to reference external memory. Useful for emulation and debug, and essential for testing |

| Symbol | Pin No. | Function   |
|--------|---------|--|
|        |         | and program verification. (Active high)  |
| XTAL1  | 2       | One side of crystal input for internal oscillator. Also input for external source. (Non TTL $V_{IH}$ ) |
| XTAL2  | 3       | Other side of crystal input.   |

### IDLE MODE DESCRIPTION

The 80C49-7, when placed into Idle mode, keeps the oscillator, the internal timer and the external interrupt and counter pins functioning and maintains the internal register and RAM status.

To place the 80C49-7 in Idle mode, a command instruction (op code 01H) is executed. To terminate Idle mode, a reset must be performed or interrupts must be enabled and an interrupt signal generated. There are two interrupt sources that can restore normal operation. One is an external signal applied to the interrupt pin. The other is from the overflow of the timer/counter. When either interrupt is invoked, the CPU is taken out of Idle mode and vectors to the interrupt's service routine address. Along with the Idle mode, the standard MCS<sup>®</sup>-48 power-down mode is still maintained.

|       |    |   |
|-------|----|---|
| RESET | 4  | Input which is used to initialize the processor. (Active low) (Non TTL $V_{IH}$ )   |
| WR    | 10 | Output strobe during a bus write. (Active low)<br>Used as write strobe to external data memory.   |
| ALE   | 11 | Address latch enable. This signal occurs once during each cycle and is useful as a clock output.<br>The negative edge of ALE strobes address info external data and program memory. |
| PSEN  | 9  | Program store enable. This output occurs only during a fetch to external program memory. (Active low)   |
| SS    | 5  | Single step input can be used in conjunction with   |

|    |    |  |
|----|----|--|
|    |    | read synchronously using the RD, WR strobes. The port can also be statically latched.<br>Contains the 8 low order program counter bits during an external program memory fetch, and receives the addressed instruction under the control of PSEN. Also contains the address and data during an external RAM data store instruction under control of ALE, RD, and WR. |
| T0 | 1  | Input pin testable using the conditional transfer instructions JTO and JNT0. T0 can be designated as a clock output using ENT0 CLK instruction.  |
| T1 | 38 | Input pin testable using the JTI and JNTI instructions.  |

Table 2. Instruction Set

| Accumulator    |                               |       |        |
|----------------|-------------------------------|-------|--------|
| Mnemonic       | Description                   | Bytes | Cycles |
| ADD A, R       | Add register to A             | 1     | 1      |
| ADD A, @R      | Add data memory to A          | 1     | 1      |
| ADD A, # data  | Add immediate to A            | 2     | 2      |
| ADDC A, R      | Add register with carry       | 1     | 1      |
| ADDC A, @R     | Add data memory with carry    | 1     | 1      |
| ADDC A, # data | Add immediate with carry      | 2     | 2      |
| ANL A, R       | And register to A             | 1     | 1      |
| ANL A, @R      | And data memory to A          | 1     | 1      |
| ANL A, # data  | And immediate to A            | 2     | 2      |
| ORL A, R       | Or register to A              | 1     | 1      |
| ORL A, @R      | Or data memory to A           | 1     | 1      |
| ORL A, # data  | Or immediate to A             | 2     | 2      |
| XRL A, R       | Exclusive or register to A    | 1     | 1      |
| XRL A, @R      | Exclusive or data memory to A | 1     | 1      |
| XRL A, # data  | Exclusive or immediate to A   | 2     | 2      |
| INC A          | Increment A                   | 1     | 1      |
| DEC A          | Decrement A                   | 1     | 1      |
| CLR A          | Clear A                       | 1     | 1      |
| CPL A          | Complement A                  | 1     | 1      |
| DA A           | Decimal adjust A              | 1     | 1      |
| SWAP A         | Swap nibbles of A             | 1     | 1      |
| RL A           | Rotate A left                 | 1     | 1      |
| RLC A          | Rotate A left through carry   | 1     | 1      |
| RR A           | Rotate A right                | 1     | 1      |
| RRC A          | Rotate A right through carry  | 1     | 1      |

| Input/Output    |                           |       |        |
|-----------------|---------------------------|-------|--------|
| Mnemonic        | Description               | Bytes | Cycles |
| IN A, P         | Input port to A           | 1     | 2      |
| OUTL P, A       | Output A to port          | 1     | 2      |
| ANL P, # data   | And immediate to port     | 2     | 2      |
| ORL P, # data   | Or immediate to port      | 2     | 2      |
| INS A, BUS      | Input BUS to A            | 1     | 2      |
| OUTL BUS, A     | Output A to BUS           | 1     | 2      |
| ANL BUS, # data | And immediate to BUS      | 2     | 2      |
| ORL BUS, # data | Or immediate to BUS       | 2     | 2      |
| MOVD A, P       | Input expander port to A  | 1     | 2      |
| MOVD P, A       | Output A to expander port | 1     | 2      |
| ANLD P, A       | And A to expander port    | 1     | 2      |
| ORLD P, A       | Or A to expander port     | 1     | 2      |

| Registers |                       |       |        |
|-----------|-----------------------|-------|--------|
| Mnemonic  | Description           | Bytes | Cycles |
| INC R     | Increment register    | 1     | 1      |
| INC @R    | Increment data memory | 1     | 1      |
| DEC R     | Decrement register    | 1     | 1      |

| Branch       |                             |       |        |
|--------------|-----------------------------|-------|--------|
| Mnemonic     | Description                 | Bytes | Cycles |
| JMP addr     | Jump unconditional          | 2     | 2      |
| JMPP @A      | Jump indirect               | 1     | 2      |
| DJNZ R, addr | Decrement register and skip | 2     | 2      |
| JC addr      | Jump on carry = 1           | 2     | 2      |
| JNC addr     | Jump on carry = 0           | 2     | 2      |
| JZ addr      | Jump on A zero              | 2     | 2      |
| JNZ addr     | Jump on A not zero          | 2     | 2      |
| JT0 addr     | Jump on T0 = 1              | 2     | 2      |
| JNT0 addr    | Jump on T0 = 0              | 2     | 2      |
| JT1 addr     | Jump on T1 = 1              | 2     | 2      |
| JNT1 addr    | Jump on T1 = 0              | 2     | 2      |
| JF0 addr     | Jump on F0 = 1              | 2     | 2      |
| JF1 addr     | Jump on F1 = 1              | 2     | 2      |
| JTF addr     | Jump on timer flag          | 2     | 2      |
| JNI addr     | Jump on INT = 0             | 2     | 2      |
| JBb addr     | Jump on accumulator bit     | 2     | 2      |

| Subroutine |                           |       |        |
|------------|---------------------------|-------|--------|
| Mnemonic   | Description               | Bytes | Cycles |
| CALL addr  | Jump to subroutine        | 2     | 2      |
| RET        | Return                    | 1     | 2      |
| RETR       | Return and restore status | 1     | 2      |

| Flags    |                   |       |        |
|----------|-------------------|-------|--------|
| Mnemonic | Description       | Bytes | Cycles |
| CLR C    | Clear carry       | 1     | 1      |
| CPL C    | Complement carry  | 1     | 1      |
| CLR F0   | Clear flag 0      | 1     | 1      |
| CPL F0   | Complement flag 0 | 1     | 1      |
| CLR F1   | Clear flag 1      | 1     | 1      |
| CPL F1   | Complement flag 1 | 1     | 1      |

Table 2. Instruction Set (Continued)

| Data Moves             |                                   |       |        | Timer/Counter   |                                 |       |        |
|------------------------|-----------------------------------|-------|--------|-----------------|---------------------------------|-------|--------|
| Mnemonic               | Description                       | Bytes | Cycles | Mnemonic        | Description                     | Bytes | Cycles |
| MOV A, R               | Move register to A                | 1     | 1      | MOV A, T        | Read timer/counter              | 1     | 1      |
| MOV A, @R              | Move data memory to A             | 1     | 1      | MOV T, A        | Load timer/counter              | 1     | 1      |
| MOV A, # data          | Move immediate to A               | 2     | 2      | STRT T          | Start timer                     | 1     | 1      |
| MOV R, A               | Move A to register                | 1     | 1      | STRT CNT        | Start counter                   | 1     | 1      |
| MOV @R, A              | Move A to data memory             | 1     | 1      | STOP TCNT       | Stop timer/counter              | 1     | 1      |
| MOV R, # data          | Move immediate to register        | 2     | 2      | EN TCNTI        | Enable timer/counter interrupt  | 1     | 1      |
| MOV @R, # data         | Move immediate to data memory     | 2     | 2      | DIS TCNTI       | Disable timer/counter interrupt | 1     | 1      |
| MOV A, PSW             | Move PSW to A                     | 1     | 1      | Control         |                                 |       |        |
| MOV PSW, A             | Move A to PSW                     | 1     | 1      | Mnemonic        | Description                     | Bytes | Cycles |
| XCH A, R               | Exchange A and register           | 1     | 1      | EN I            | Enable external interrupt       | 1     | 1      |
| XCH A, @R              | Exchange A and data memory        | 1     | 1      | DIS I           | Disable external interrupt      | 1     | 1      |
| XCHD A, @R             | Exchange nibble of A and register | 1     | 1      | SEL RB0         | Select register bank 0          | 1     | 1      |
| MOVX A, @R             | Move external data memory to A    | 1     | 2      | SEL RB1         | Select register bank 1          | 1     | 1      |
| MOVX @R, A             | Move A to external data memory    | 1     | 2      | SEL MB0         | Select memory bank 0            | 1     | 1      |
| MOVP A, @A             | Move to A from current page       | 1     | 2      | SEL MB1         | Select memory bank 1            | 1     | 1      |
| MOVP3 A, @A            | Move to A from page 3             | 1     | 2      | ENTO CLK        | Enable clock output on T0       | 1     | 1      |
| Data Moves (Continued) |                                   |       |        | Mnemonic        | Description                     | Bytes | Cycles |
| RET                    | Return                            | 1     | 2      | NOP             | No operation                    | 1     | 1      |
| PTR                    | Return and restore status         | 1     | 2      | IDL             | Select Idle Operation           | 1     | 1      |
| Flags                  |                                   |       |        | Input/Output    |                                 |       |        |
| Mnemonic               | Description                       | Bytes | Cycles | Mnemonic        | Description                     | Bytes | Cycles |
| CLR C                  | Clear carry                       | 1     | 1      | IN A, P         | Input port to A                 | 1     | 2      |
| CPL C                  | Complement carry                  | 1     | 1      | OUT A, P        | Output A to port                | 1     | 2      |
| CLR F0                 | Clear flag 0                      | 1     | 1      | ANL A, # data   | And immediate to port           | 2     | 2      |
| CPL F0                 | Complement flag 0                 | 1     | 1      | ORL A, # data   | Or immediate to port            | 2     | 2      |
| CLR F1                 | Clear flag 1                      | 1     | 1      | IN A, BUS       | Input BUS to A                  | 1     | 2      |
| CPL F1                 | Complement flag 1                 | 1     | 1      | OUT A, BUS      | Output A to BUS                 | 1     | 2      |
| CLR F2                 | Clear flag 2                      | 1     | 1      | ANL BUS, # data | And immediate to BUS            | 2     | 2      |
| CPL F2                 | Complement flag 2                 | 1     | 1      | ORL BUS, # data | Or immediate to BUS             | 2     | 2      |
| CLR F3                 | Clear flag 3                      | 1     | 1      | IN A, # data    | Input # data to A               | 1     | 2      |
| CPL F3                 | Complement flag 3                 | 1     | 1      | OUT A, # data   | Output A to # data              | 1     | 2      |
| CLR F4                 | Clear flag 4                      | 1     | 1      | ANL A, A        | And A to A                      | 1     | 1      |
| CPL F4                 | Complement flag 4                 | 1     | 1      | ORL A, A        | Or A to A                       | 1     | 1      |
| CLR F5                 | Clear flag 5                      | 1     | 1      | ANL A, R        | And register to A               | 1     | 1      |
| CPL F5                 | Complement flag 5                 | 1     | 1      | ORL A, R        | Or register to A                | 1     | 1      |
| CLR F6                 | Clear flag 6                      | 1     | 1      | ANL A, @R       | And data memory to A            | 1     | 1      |
| CPL F6                 | Complement flag 6                 | 1     | 1      | ORL A, @R       | Or data memory to A             | 1     | 1      |
| CLR F7                 | Clear flag 7                      | 1     | 1      | ANL A, # data   | And immediate to A              | 2     | 2      |
| CPL F7                 | Complement flag 7                 | 1     | 1      | ORL A, # data   | Or immediate to A               | 2     | 2      |

**ABSOLUTE MAXIMUM RATINGS\***

Ambient Temperature Under Bias . . . . . 0°C to 70°C  
 Storage Temperature . . . . . -65°C to +150°C  
 Voltage On Any Pin With Respect to Ground . . . . . -0.5V to V<sub>CC</sub>+1V  
 Maximum Voltage On Any Pin With Respect to Ground . . . . . 7V  
 Power Dissipation . . . . . 1.0 Watt

*\*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of device at these or any other conditions above those indicated in the operational sections of this specification is not implied.*

**D.C. CHARACTERISTICS:** ( $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ;  $V_{CC} = V_{DD} = 5\text{V} \pm 20\%$ ;  $|V_{CC} - V_{DD}| \leq 1.5\text{V}$ ;  $V_{SS} = 0\text{V}$ )

| Symbol    | Parameter  | Limits             |     |              | Unit          | Test Conditions  |
|-----------|--|--------------------|-----|--------------|---------------|--|
|           |  | Min                | Typ | Max          |               |  |
| $V_{IL}$  | Input Low Voltage<br>(All Except X1, RESET)                | .5                 |     | .18 $V_{CC}$ | V             |  |
| $V_{IL1}$ | Input Low Voltage X1, RESET                                | 5                  |     | .13 $V_{CC}$ | V             |  |
| $V_{IH}$  | Input High Voltage<br>(All Except XTAL1, RESET)            | $0.2 V_{CC} + 1.2$ |     | $V_{CC}$     | V             |  |
| $V_{IH1}$ | Input High Voltage (X1, RESET)                             | .7 $V_{CC}$        |     | $V_{CC}$     | V             |  |
| $V_{OL}$  | Output Low Voltage (BUS)                                   |                    |     | .6           | V             | $I_{OL} = 2.0\text{ mA}$   |
| $V_{OL1}$ | Output Low Voltage<br>(RD, WR, PSEN, ALE)                  |                    |     | .6           | V             | $I_{OL} = 1.8\text{ mA}$   |
| $V_{OL2}$ | Output Low Voltage (PROG)                                  |                    |     | .6           | V             | $I_{OL} = 1.0\text{ mA}$   |
| $V_{OL3}$ | Output Low Voltage<br>(All Other Outputs)                  |                    |     | .6           | V             | $I_{OL} = 1.6\text{ mA}$   |
| $V_{OH}$  | Output High Voltage (BUS)                                  | .75 $V_{CC}$       |     |              | V             | $I_{OH} = -400\text{ }\mu\text{A}$                                     |
| $V_{OH1}$ | Output High Voltage<br>(RD, WR, PSEN, ALE)                 | .75 $V_{CC}$       |     |              | V             | $I_{OH} = -100\text{ }\mu\text{A}$                                     |
| $V_{OH2}$ | Output High Voltage<br>(All Other Outputs)                 | 2.4<br>3.0         |     |              | V             | $I_{OH} = -40\text{ }\mu\text{A}$<br>$I_{OH} = -20\text{ }\mu\text{A}$ |
| $I_{L1}$  | Input Leakage Current (T1, INT, EA)                        |                    |     | $\pm 5$      | $\mu\text{A}$ | $V_{SS} \leq V_{IN} \leq V_{CC}$                                       |
| $I_{L11}$ | Input Leakage Current<br>(P10-P17, P20-P27, SS)            |                    |     | -500         | $\mu\text{A}$ | $V_{SS} \leq V_{IN} \leq V_{CC}$                                       |
| $I_{LO}$  | Output Leakage Current (BUS, TO)<br>(High Impedance State) |                    |     | $\pm 5$      | $\mu\text{A}$ | $V_{SS} \leq V_{IN} \leq V_{CC}$                                       |
| $I_{LR}$  | Input Leakage Current (RESET)                              | -10                |     | -300         | $\mu\text{A}$ | $V_{SS} \leq V_{IN} \leq V_{IH1}$                                      |
| $I_{PD}$  | Power Down Standby Current                                 |                    |     | 2            | $\mu\text{A}$ | $V_{DD} = 2.0\text{V}$ RESET $\leq V_{IL}$                             |

**$I_{CC}$  Active Current (mA)**

| $V_{CC}$ | 4V  | 5V  | 6V  |
|----------|-----|-----|-----|
| 1 MHz    | 2.5 | 3.3 | 4.0 |
| 6 MHz    | 5   | 6.8 | 8.5 |
| 11 MHz   | 9   | 12  | 15  |

**$I_{CC}$  Idle Current (mA)**

| $V_{CC}$ | 4V  | 5V  | 6V  |
|----------|-----|-----|-----|
| 1 MHz    | 1.7 | 2.0 | 2.2 |
| 6 MHz    | 2   | 3   | 4   |
| 11 MHz   | 3.5 | 4.8 | 6   |

**Absolute Maximum Unloaded Current**

**$I_{CC}$  Test Conditions:**

**$I_{CC}$  Active**

All outputs disconnected  
T1, INT, SS, T0 connected to HIGH ( $V_{IH}$ )  
EA, RST connected to LOW ( $V_{IL}$ )  
XTAL1 External Drive  
Rise Time = 10 ns, Fall Time = 10 ns  
XTAL2 No connection  
 $V_{IH} = V_{CC} - 0.5\text{V}$   
 $V_{IL} = V_{SS} + 0.5\text{V}$

**$I_{CC}$  Idle**

All outputs disconnected  
XTAL1 External Drive  
Rise Time = 10 ns, Fall Time = 10 ns  
XTAL2 No connection  
 $V_{IH} = V_{CC} - 0.5\text{V}$   
 $V_{IL} = V_{SS} + 0.5\text{V}$



**A.C. CHARACTERISTICS:** ( $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ;  $V_{CC} = V_{DD} = 5V \pm 20\%$ ;  $|V_{CC} - V_{DD}| \leq 1.5V$ ;  $V_{SS} = 0V$ )

| Symbol             | Parameter  | f (t)<br>(Note 3) | 11 MHz |      | Unit          | Conditions<br>(Note 1) |
|--------------------|--|-------------------|--------|------|---------------|------------------------|
|                    |  |                   | Min    | Max  |               |                        |
| t                  | Clock Period   | 1/xtal freq       | 90.9   | 1000 | ns            | (Note 3)               |
| t <sub>LL</sub>    | ALE Pulse Width  | 3.5t-170          | 150    |      | ns            |                        |
| t <sub>AL</sub>    | Addr Setup to ALE  | 2t-110            | 70     |      | ns            | (Note 2)               |
| t <sub>LA</sub>    | Addr Hold from ALE   | t-40              | 50     |      | ns            |                        |
| t <sub>CC1</sub>   | Control Pulse Width ( $\overline{RD}$ , $\overline{WR}$ )                | 7.5t-200          | 480    |      | ns            |                        |
| t <sub>CC2</sub>   | Control Pulse Width ( $\overline{PSEN}$ )                                | 6t-200            | 350    |      | ns            |                        |
| t <sub>DW</sub>    | Data Setup before $\overline{WR}$  | 6.5t-200          | 390    |      | ns            |                        |
| t <sub>WD</sub>    | Data Hold after $\overline{WR}$  | t-50              | 40     |      | ns            |                        |
| t <sub>DR</sub>    | Data Hold ( $\overline{RD}$ , $\overline{PSEN}$ )                        | 1.5t-30           | 0      | 110  | ns            |                        |
| t <sub>RD1</sub>   | $\overline{RD}$ to Data in   | 6t-170            |        | 350  | ns            |                        |
| t <sub>RD2</sub>   | $\overline{PSEN}$ to Data in   | 4.5t-170          |        | 190  | ns            |                        |
| t <sub>AW</sub>    | Addr Setup to $\overline{WR}$  | 5t-150            | 300    |      | ns            |                        |
| t <sub>AD1</sub>   | Addr Setup to Data ( $\overline{RD}$ )                                   | 10.5t-220         |        | 730  | ns            |                        |
| t <sub>AD2</sub>   | Addr Setup to Data ( $\overline{PSEN}$ )                                 | 7.5t-220          |        | 460  | ns            |                        |
| t <sub>AFC1</sub>  | Addr Float to $\overline{RD}$ , $\overline{WR}$                          | 2t-40             | 140    |      | ns            | (Note 2)               |
| t <sub>AFC2</sub>  | Addr Float to $\overline{PSEN}$  | .5t-40            | 10     |      | ns            | (Note 2)               |
| t <sub>LAFC1</sub> | ALE to Control ( $\overline{RD}$ , $\overline{WR}$ )                     | 3t-75             | 200    |      | ns            |                        |
| t <sub>LAFC2</sub> | ALE to Control ( $\overline{PSEN}$ )                                     | 1.5t-75           | 60     |      | ns            |                        |
| t <sub>CA1</sub>   | Control to ALE ( $\overline{RD}$ , $\overline{WR}$ , $\overline{PROG}$ ) | t-65              | 25     |      | ns            |                        |
| t <sub>CA2</sub>   | Control to ALE ( $\overline{PSEN}$ )                                     | 4t-70             | 290    |      | ns            |                        |
| t <sub>CP</sub>    | Port Control Setup to $\overline{PROG}$                                  | 1.5t-80           | 50     |      | ns            |                        |
| t <sub>PC</sub>    | Port Control Hold to $\overline{PROG}$                                   | 4t-260            | 100    |      | ns            |                        |
| t <sub>PR</sub>    | $\overline{PROG}$ to P2 Input Valid                                      | 8.5t-120          |        | 650  | ns            |                        |
| t <sub>PF</sub>    | Input Data Hold from $\overline{PROG}$                                   | 1.5t              | 0      | 140  | ns            |                        |
| t <sub>DP</sub>    | Output Data Setup  | 6t-290            | 250    |      | ns            |                        |
| t <sub>PD</sub>    | Output Data Hold   | 1.5t-90           | 40     |      | ns            |                        |
| t <sub>PP</sub>    | $\overline{PROG}$ Pulse Width  | 10.5t-250         | 700    |      | ns            |                        |
| t <sub>PL</sub>    | Port 2 I/O Setup to ALE  | 4t-200            | 160    |      | ns            |                        |
| t <sub>LP</sub>    | Port 2 I/O Hold to ALE   | 1.5t-120          | 15     |      | ns            |                        |
| t <sub>pv</sub>    | Port Output from ALE   | 4.5t+100          |        | 510  | ns            |                        |
| t <sub>OPRR</sub>  | T0 Rep Rate  | 3t                | 270    |      | ns            |                        |
| t <sub>CY</sub>    | Cycle Time   | 15t               | 1.36   | 15.0 | $\mu\text{s}$ |                        |

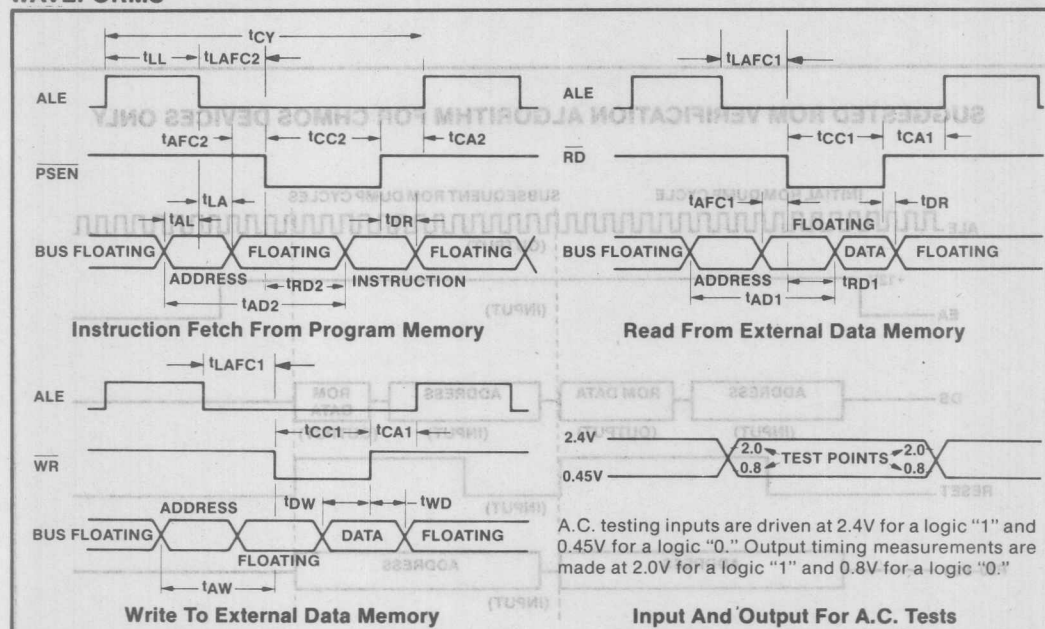
**Notes:**

1. Control Outputs CL = 80pF  
BUS Outputs CL = 150pF

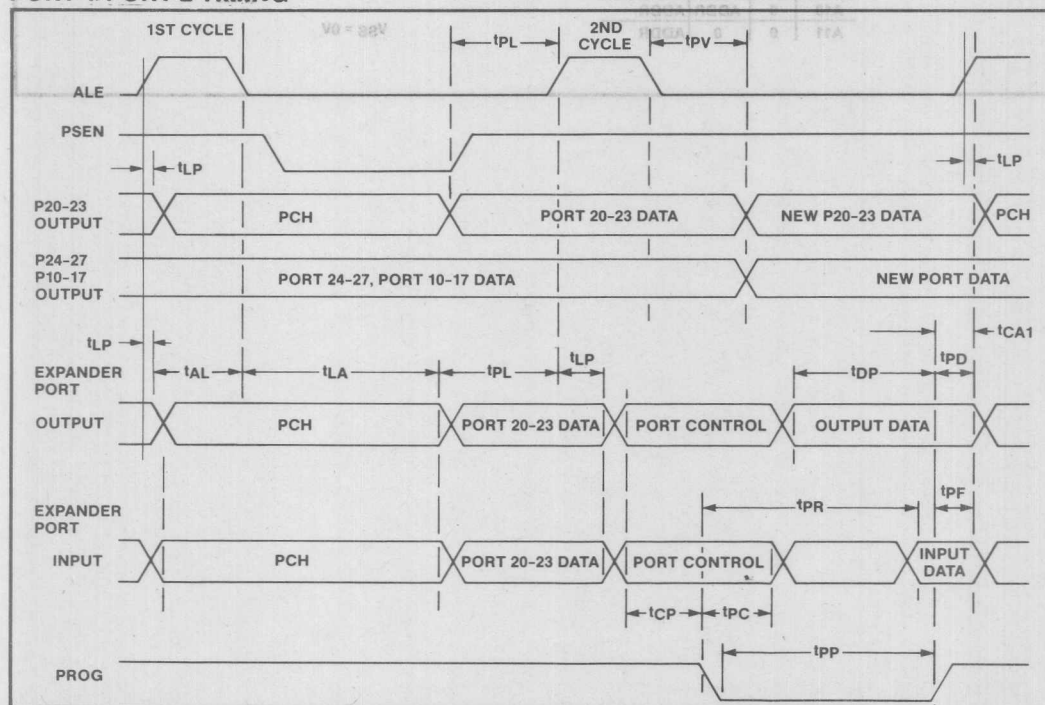
2. BUS High Impedance  
Load 20pF

3. f(t) assumes 50% duty cycle on X1, X2. Max  
clock period is for a 1 MHz crystal input.

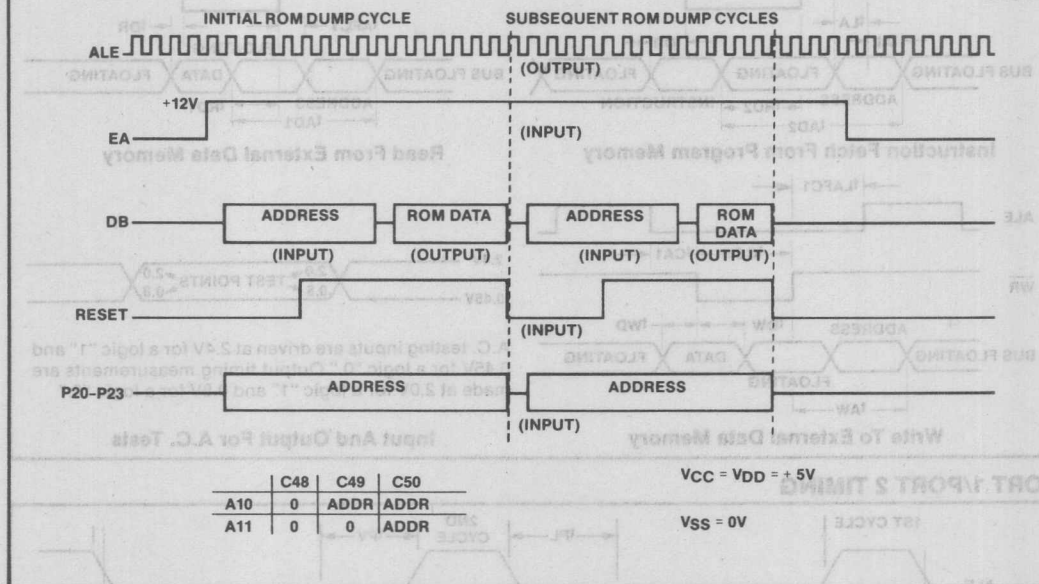
# WAVEFORMS



## PORT 1/PORT 2 TIMING



# SUGGESTED ROM VERIFICATION ALGORITHM FOR CHMOS DEVICES ONLY







16

The RUP<sup>TM</sup>-44 Family

---

## CHAPTER 16

### THE RUPI™-44 FAMILY: MICROCONTROLLER WITH ON-CHIP COMMUNICATION CONTROLLER

#### 16.0 INTRODUCTION

The RUPI-44 family is designed for applications requiring local intelligence at remote nodes, and communication capability among these distributed nodes. The RUPI-44 integrates onto a single chip Intel's highest performance microcontroller, the 8051-core, with an intelligent and high performance Serial communication controller, called the Serial Interface Unit, or SIU. See Figure 16-1. This dual controller architecture allows complex control and high speed data communication functions to be realized cost effectively.

The RUPI-44 family consists of three pin compatible parts:

- 8344—8051 Microcontroller with SIU
- 8044—An 8344 with 4K bytes of on-chip ROM program memory.
- 8744—An 8344 with 4K bytes of on-chip EPROM program memory.

#### 16.1 ARCHITECTURE OVERVIEW

The 8044's dual controller architecture enables the RUPI to perform complex control tasks and high speed communication in a distributed network environment.

The 8044 microcontroller is the 8051-core, and maintains complete software compatibility with it. The microcontroller contains a powerful CPU with on-chip peripherals, making it capable of serving sophisticated

real-time control applications such as instrumentation, industrial control, and intelligent computer peripherals. The microcontroller features on-chip peripherals such as two 16-bit timer/counters and 5 source interrupt capability with programmable priority levels. The microcontroller's high performance CPU executes most instructions in 1 microsecond, and can perform an  $8 \times 8$  multiply in 4 microseconds. The CPU features a Boolean processor that can perform operations on 256 directly addressable bits. 192 bytes of on-chip data RAM can be extended to 64K bytes externally. 4K bytes of on-chip program ROM can be extended to 64K bytes externally. The CPU and SIU run concurrently. See Figure 16-2.

The SIU is designed to perform serial communications with little or no CPU involvement. The SIU supports data rates up to 2.4Mbps, externally clocked, and 375K bps self clocked (i.e., the data clock is recovered by an on-chip digital phase locked loop). SIU hardware supports the HDLC/SDLC protocol: zero bit insertion/deletion, address recognition, cyclic redundancy check, and frame number sequence check are automatically performed.

The SIU's Auto mode greatly reduces communication software overhead. The AUTO mode supports the SDLC Normal Response Mode, by performing secondary station responses in hardware without any CPU involvement. The Auto mode's interrupt control and frame sequence numbering capability eliminates software overhead normally required in conventional systems. By using the Auto mode, the CPU is free to concentrate on real time control of the application.

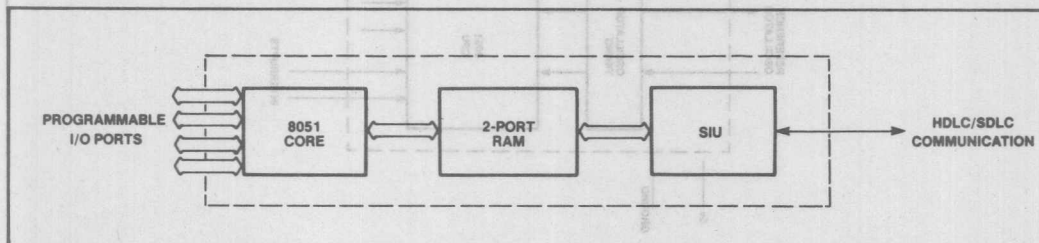
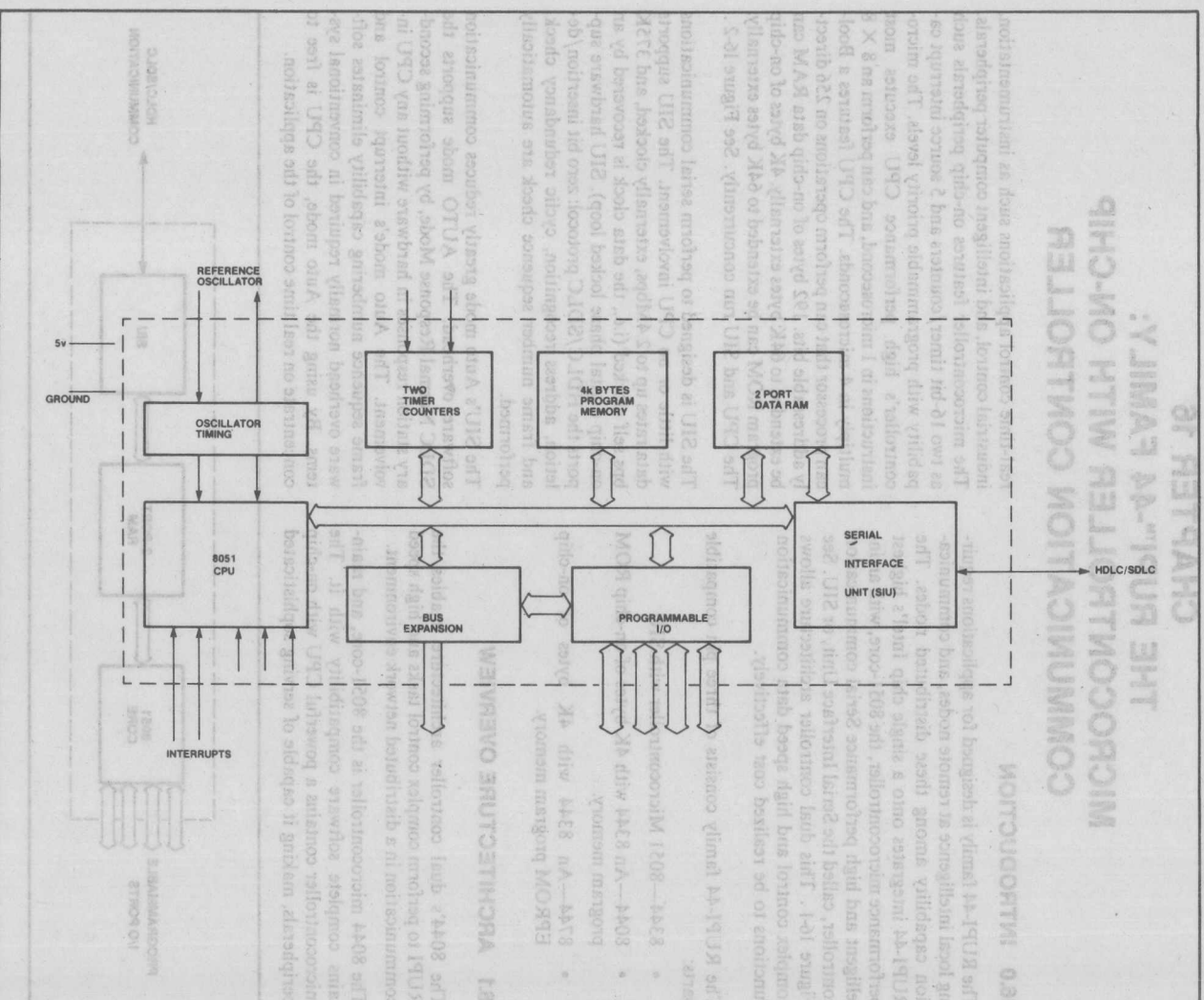


Figure 16-1. RUPI™-44 Dual Controller Architecture

COMMUNICATION CONTROLLER  
MICROCONTROLLER WITH ON-CHIP  
HDL/C/SOLC  
THE BUS-44 FAMILY: 80 PETAHZ



**Figure 16-2. Simplified 8044 Block Diagram**

## 16.2 THE HDLC/SDLC PROTOCOLS

### 16.2.1 HDLC/SDLC Advantages over Async

The High Level Data Link Control, HDLC, is a standard communication link control established by the International Standards Organization (ISO). SDLC is a subset of HDLC.

HDLC and SDLC are both well recognized standard serial protocols. The Synchronous Data Link Control, SDLC, is an IBM standard communication protocol. IBM originally developed SDLC to provide efficient, reliable and simple communication between terminals and computers.

The major advantages of SDLC/HDLC over Asynchronous communications protocol (Async):

- **SIMPLE:** Data Transparency

- **EFFICIENT:** Well Defined Message-Level Operation
- **RELIABLE:** Frame Check Sequence and Frame Numbering

The SDLC reduces system complexity. HDLC/SDLC are "data transparent" protocols. Data transparency means that an arbitrary data stream can be sent without concern that some of data could be mistaken for a protocol controller. Data transparency relieves the communication controller having to detect special characters.

SDLC/HDLC provides more data throughput than Async. SDLC/HDLC runs at Message-level Operation which transmits multiple bytes within the frame. Whereas Async is based on character-level operation. Async transmits or receives a character at a time. Since Async requires start and stop bits in every transmission, there is a considerable waste of overhead compared to SDLC/HDLC.

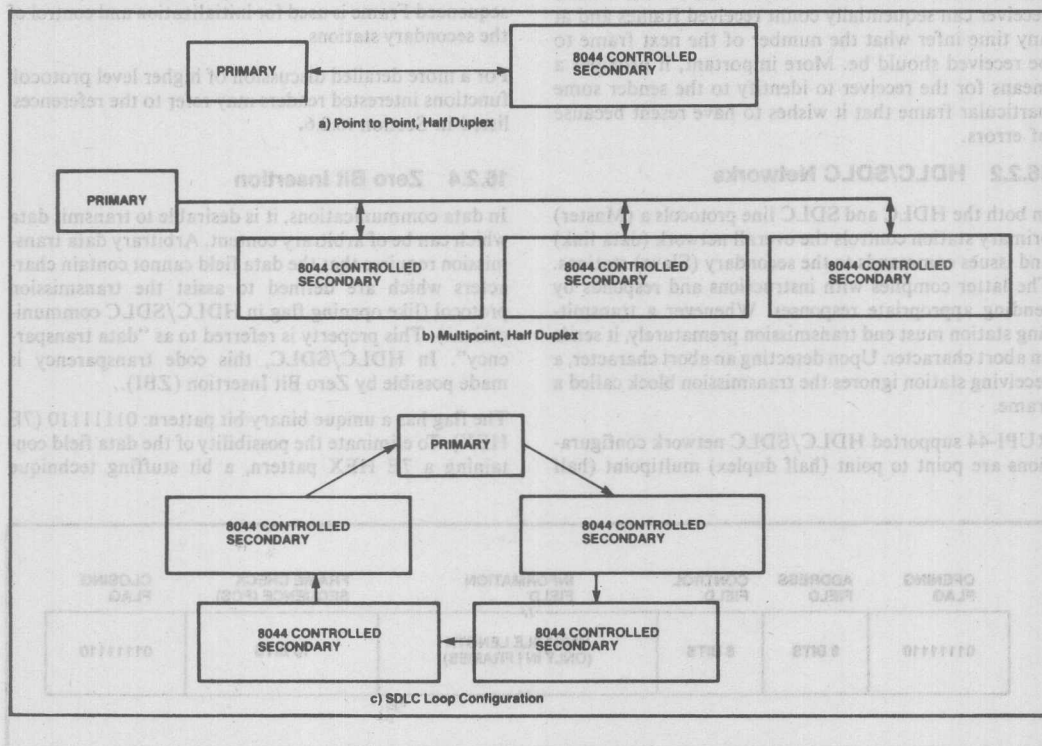


Figure 16-3. RUP™-44 Supported Network Configurations



Due to SDLC/HDLC's well delineated field (see Figure 16-4) the CPU does not have to interpret character by character to determine control field and information field. In the case of Async, CPU must look at each character to interpret what it means. The practical advantage of such feature is straight forward use of DMA for information transfer.

In addition, SDLC/HDLC further improves Data throughput using implied Acknowledgement of transferred information. A station using SDLC/HDLC may acknowledge previously received information while transmitting different information in the same frame. In addition, up to 7 messages may be outstanding before an acknowledgement is required.

The HDLC/SDLC protocol can be used to realize reliable data links. Reliable Data transmission is ensured at the bit level by sending a frame check sequence, cyclic redundancy checking, within the frame. Reliable frame transmission is ensured by sending a frame number identification with each frame. This means that a receiver can sequentially count received frames and at any time infer what the number of the next frame to be received should be. More important, it provides a means for the receiver to identify to the sender some particular frame that it wishes to have resent because of errors.

### 16.2.2 HDLC/SDLC Networks

In both the HDLC and SDLC line protocols a (Master) primary station controls the overall network (data link) and issues commands to the secondary (Slave) stations. The latter complies with instructions and responds by sending appropriate responses. Whenever a transmitting station must end transmission prematurely, it sends an abort character. Upon detecting an abort character, a receiving station ignores the transmission block called a frame.

RUP1-44 supported HDLC/SDLC network configurations are point to point (half duplex) multipoint (half

duplex), and loop. In the loop configuration the stations themselves act as repeaters, so that long links can be easily realized, see Figure 16-3.

### 16.2.3 Frames

An HDLC/SDLC frame consists of five basic fields: Flag, Address, Control, Data and Error Detection. A frame is bounded by flags—opening and closing flags. An address field is 8 bits wide in SDLC, extendable to 2 or more bytes in HDLC. The control field is also 8 bits wide, extendable to two bytes in HDLC. The SDLC data field or information field may be any number of bytes. The HDLC data field may or may not be on an 8 bit boundary. A powerful error detection code called Frame Check Sequence contains the calculated CRC (Cycle Redundancy Code) for all the bits between the flags. See Figure 16-4.

In HDLC and SDLC are three types of frames; an Information Frame is used to transfer data, a Supervisory Frame is used for control purposes, and a Non-sequenced Frame is used for initialization and control of the secondary stations.

For a more detailed discussion of higher level protocol functions interested readers may refer to the references listed in Section 16.2.6.

### 16.2.4 Zero Bit Insertion

In data communications, it is desirable to transmit data which can be of arbitrary content. Arbitrary data transmission requires that the data field cannot contain characters which are defined to assist the transmission protocol (like opening flag in HDLC/SDLC communications). This property is referred to as "data transparency". In HDLC/SDLC, this code transparency is made possible by Zero Bit Insertion (ZBI).

The flag has a unique binary bit pattern: 01111110 (7E HEX). To eliminate the possibility of the data field containing a 7E HEX pattern, a bit stuffing technique

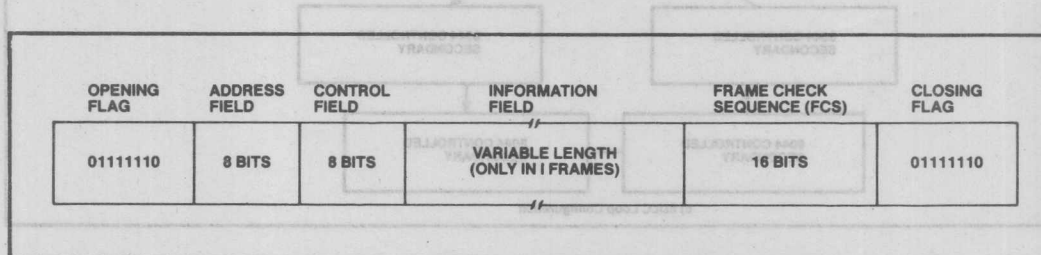


Figure 16-4. Frame Format

called Zero Bit Insertion is used. This technique specifies that during transmission, a binary 0 be inserted by the transmitter after any succession of five contiguous binary 1's. This will ensure that no pattern of 0111110 is ever transmitted between flags. On the receiving side, after receiving the flag, the receiver hardware automatically deletes any 0 following five consecutive 1's. The 8044 performs zero bit insertion and deletion automatically.

### 16.2.5 Non-return to Zero Inverted (NRZI)

NRZI is a method of clock and data encoding that is well suited to the HDLC/SDLC protocol. It allows HDLC/SDLC protocols to be used with low cost asynchronous modems. NRZI coding is done at the transmitter to enable clock recovery from the data at the receiver terminal by using standard digital phase locked loop (DPLL) techniques. NRZI coding specifies that the signal condition does not change for transmitting a 1, while an 0 causes a change of state. NRZI coding ensures that an active data line will have a transition at least every 5-bit times (recall Zero Bit Insertion), while contiguous 0's will cause a change of state. Thus, ZBI and NRZI encoding makes it possible for the 8044's on-chip DPLL to recover a receive clock (from received data) synchronized to the received data and at the same time ensure data transparency.

### 16.2.6 References

1. IBM Synchronous Data Link Control General Information GA27-3093-2 File No. GENL-09.
2. Standard Network Access Protocol Specification, DATAPAC Trans-Canada Telephone System CCG111.
3. IBM 3650 Retail Store System Loop Interface OEM Information, IBM, GA27-3098-0
4. Guidebook to Data Communications, Training Manual, Hewlett-Packard 5955-1715
5. "Serial Backplane Suits Multiprocessor Architectures", Mike Webb, *Computer Design*, July 1984, p. 85-96.
6. "Serial Bus Simplifies Distributed Control", P.D. MacWilliams, *Control Engineering*, June 1984, p. 101-104.
7. "Chips Support Two Local Area Networks", Bob Dahlberg, *Computer Design*, May 1984, p. 107-114.
8. "Build a VLSI-based Workstation for the Ethernet Environment", Mike Webb, *EDN*, 23 February 1984, p. 297-307.
9. "Networking With the 8044", Young Sohn & Charles Gopen, *Digital Design*, May 1984, p. 136-137.

## 16.3 RUP™-44 DESIGN SUPPORT

### 16.3.1 Design Tool Support

A critical design consideration is time to market. Intel provides a sophisticated set of design tools to speed hardware and software development time of 8044 based products. These include ICE-44, ASM-51, PL/M-51, and EMV-44.



**Figure 16-5. RUP™-44 Development Support Configuration Intellec® System, ICE™-44 Buffer Box, and ICE-44 Module Plugged into a User Prototype Board.**

A primary tool is the 8044 In Circuit Emulator, called ICE-44. See Figure 16-5. In conjunction with Intel's Intellec® Microprocessor Development System, the ICE-44 emulator allows hardware and software development to proceed interactively. This approach is more effective than the traditional method of independent hardware and software development followed by system integration. With the ICE-44 module, prototype hardware can be added to the system as it is designed. Software and hardware integration occurs while the product is being developed.

The ICE-44 emulator assists four stages of development:

#### 1) Software Debugging

It can be operated without being connected to the user's system before any of the user's hardware is available. In this stage ICE-44 debugging capabilities can be used in conjunction with the Intellec text edi-

tor and 8044 macroassembler to facilitate program development.

## 2) Hardware Development

The ICE-44 module's precise emulation characteristics and full-speed program RAM make it a valuable tool for debugging hardware, including the time-critical SDLC serial port, parallel port, and timer interfaces.

## 3) System Integration

Integration of software and hardware can begin when any functional element of the user system hardware is connected to the 8044 socket. As each section of the user's hardware is completed, it is added to the prototype. Thus, each section of the hardware and software is system tested in real-time operation as it becomes available.

## 4) System Test

When the user's prototype is complete, it is tested with the final version of the user system software. The ICE-44 module is then used for real-time emulation of the 8044 to debug the system as a completed unit.

The final product verification test may be performed using the 8744 EPROM version of the 8044 microcomputer. Thus, the ICE-44 module provides the user with the ability to debug a prototype or production system at any stage in its development.

A conversion kit, ICE-44 CON, is available to upgrade an ICE-51 module to ICE-44.

Intel's ASM-51 Assembler supports the 8044 special function registers and assembly program development. PL/M-51 provides designers with a high level language for the 8044. Programming in PL/M can greatly reduce development time, and ensure quick time to market.

These tools have recently been expanded with the addition of the EMV-44CON. This conversion kit allows you to convert an EMV-51 into an EMV-44 emulation vehicle. The resultant low cost emulator is design for use

with an iPDS Personal Development System, which also supports the ASM-51 assembler and PL/M-51. See Figure 16-6.



**Figure 16-6. RUP1-44 IPDS Personal Development System, EMV-44 Buffer Box, and EMV-44 Module Plugged Into a User Prototype Board.**

Emulation support is similar to the ICE-44 with support for Software and Hardware Development, System Integration, and System Test. The iPDS's rugged portability and ease of use also make it an ideal system for production tests and field service of your finished design. In addition, the iPDS offers EPROM programming module for the 8744, and direct communications with the 8044-based BITBUS via an optional iSBX344 distributed control module.

## 16.3.2 8051 Workshop

Intel provides 8051 training to its customers through the 5-day 8051 workshop. Familiarity with the 8051 and 8044 is achieved through a combination of lecture and laboratory exercises.

For designers not familiar with the 8051, the workshop is an effective way to become proficient with the 8051 architecture and capabilities.







## CHAPTER 17

# 8044 ARCHITECTURE

### 17.0 GENERAL

The 8044 is based on the 8051 core. The 8044 replaces the 8051's serial port with an intelligent HDLC/SDLC controller called the Serial Interface or SIU. Thus the differences between the two result from the 8044's increased on-chip RAM (192 bytes) and additional special function registers necessary to control the SIU. Aside from the increased memory, the SIU itself, and differences in 5 pins (for the serial port), the 8044 and 8051 are compatible.

This chapter describes the differences between the 8044 and 8051. Information pertaining to the 8051 core, eg. instruction set, port operation, EPROM programming, etc. is located in the 8051 sections of this manual.

A block diagram of the 8044 is shown in Figure 17-1. The pinpoint is shown on the inside front cover.

### 17.1 MEMORY ORGANIZATION OVERVIEW

The 8044 maintains separate address spaces for Program Memory and Data Memory. The Program Memory can be up to 64K bytes long, of which the lowest 4K bytes are in the on-chip ROM.

If the  $\overline{EA}$  pin is held high, the 8044 executes out of internal ROM unless the Program Counter exceeds 0FFFH. Fetches from locations 1000H through FFFFH are directed to external Program Memory.

If the  $\overline{EA}$  pin is held low, the 8044 fetches all instructions from external Program Memory.

The Data Memory consists of 192 bytes of on-chip RAM, plus 35 Special Function Registers, in addition to which the device is capable of accessing up to 64K bytes of external data memory.

The Program Memory uses 16-bit addresses. The external Data Memory can use either 8-bit or 16-bit addresses. The internal Data Memory uses 8-bit addresses, which provide a 256-location address space. The lower 192 addresses access the on-chip RAM. The Special Function Registers occupy various locations in the upper 128 bytes of the same address space.

The lowest 32 bytes in the internal RAM (locations 00 through 1FH) are divided into 4 banks of registers, each bank consisting of 8 bytes. Any one of these banks can be selected to be the "working registers" of the CPU, and can be accessed by a 3-bit address in the same byte as the opcode of an instruction. Thus, a large number of instructions are one-byte instructions.

The next higher 16 bytes of the internal RAM (locations 20H through 2FH) have individually addressable bits. These are provided for use as software flags or for one-bit (Boolean) processing. This bit-addressing capability is an important feature of the 8044. In addition to the 128 individually addressable bits in RAM, twelve of the Special Function Registers also have individually addressable bits.

A memory map is shown in Figure 17-2.

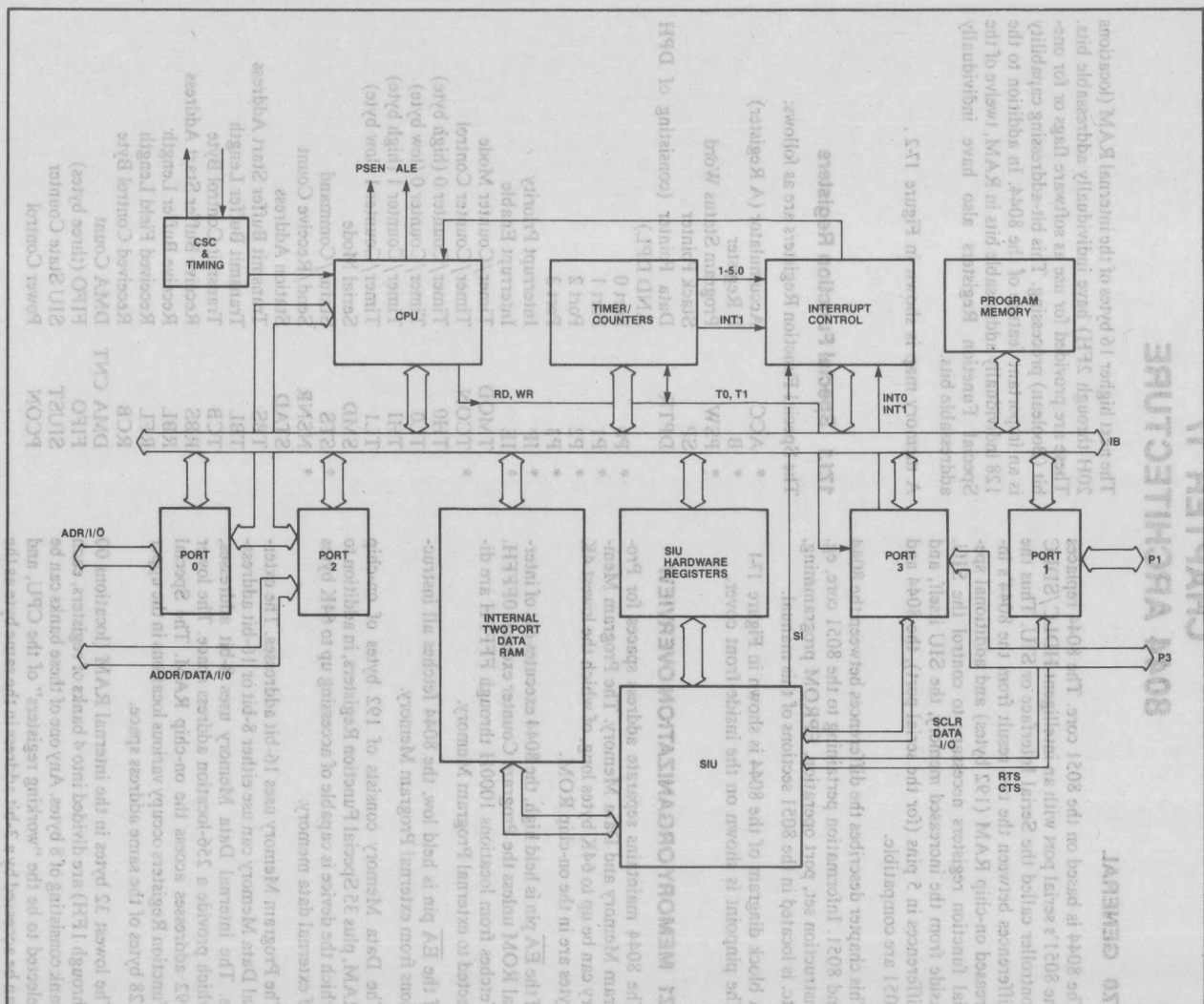
#### 17.1.1 Special Function Registers

The Special Function Registers are as follows:

|         |  |
|---------|--|
| * ACC   | Accumulator (A Register)                 |
| * B     | B Register                               |
| * PSW   | Program Status Word                      |
| SP      | Stack Pointer                            |
| DPTR    | Data Pointer (consisting of DPH AND DPL) |
| * P0    | Port 0                                   |
| * P1    | Port 1                                   |
| * P2    | Port 2                                   |
| * P3    | Port 3                                   |
| * IP    | Interrupt Priority                       |
| * IE    | Interrupt Enable                         |
| TMOD    | Timer/Counter Mode                       |
| * TCON  | Timer/Counter Control                    |
| TH0     | Timer/Counter 0 (high byte)              |
| TL0     | Timer/Counter 0 (low byte)               |
| TH1     | Timer/Counter 1 (high byte)              |
| TL1     | Timer/Counter 1 (low byte)               |
| SMD     | Serial Mode                              |
| * STS   | Status/Command                           |
| * NSNR  | Send/Receive Count                       |
| STAD    | Station Address                          |
| TBS     | Transmit Buffer Start Address            |
| TBL     | Transmit Buffer Length                   |
| TCB     | Transmit Control Byte                    |
| RBS     | Receive Buffer Start Address             |
| RBL     | Receive Buffer Length                    |
| RFL     | Received Field Length                    |
| RCB     | Received Control Byte                    |
| DMA CNT | DMA Count                                |
| FIFO    | FIFO (three bytes)                       |
| SIUST   | SIU State Counter                        |
| PCON    | Power Control                            |

The registers marked with \* are both byte- and bit-addressable.

**Figure 17-1. RUP™ Block Diagram**



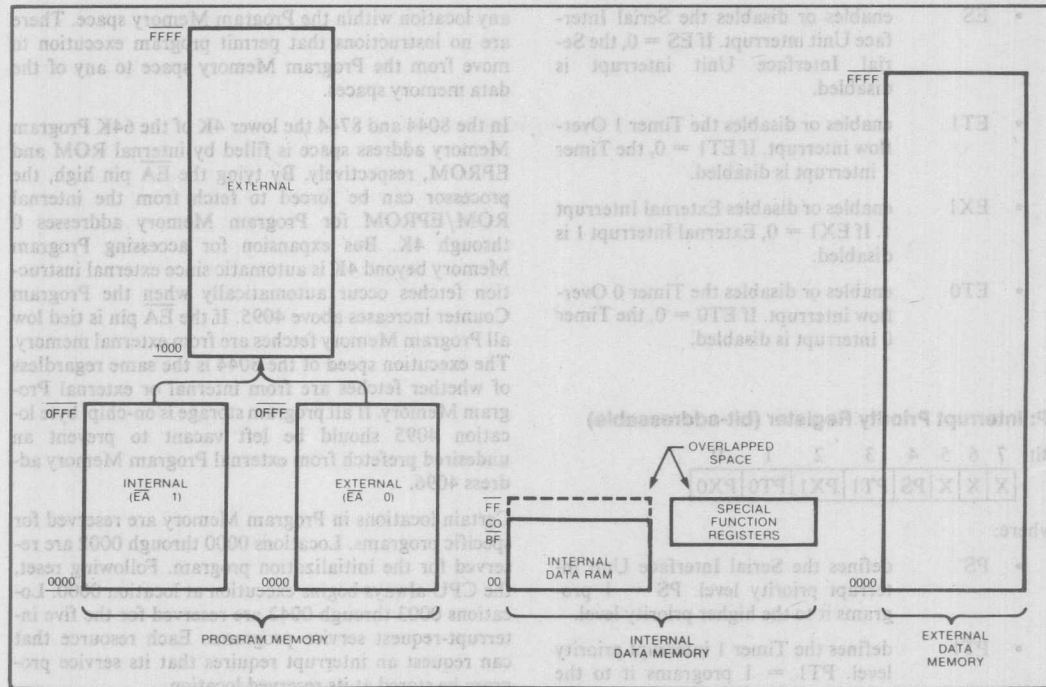


Figure 17-2: RUPITM-44 Memory Map

### Stack Pointer

The Stack Pointer is 8 bits wide. The stack can reside anywhere in the 192 bytes of on-chip RAM. When the 8044 is reset, the stack pointer is initialized to 07H. When executing a PUSH or a CALL, the stack pointer is incremented before data is stored, so the stack would begin at location 08H.

### 17.1.2 Interrupt Control Registers

The Interrupt Request Flags are as listed below:

| Source                   | Request Flag                        | Location       |
|--------------------------|-------------------------------------|----------------|
| External Interrupt 0     | INT0, if IT0 = 0<br>IE0, if IT0 = 1 | P3.2<br>TCON.1 |
| Timer 0 Overflow         | TF0                                 | TCON.5         |
| External Interrupt 1     | INT1, if IT1 = 0<br>IE1, if IT1 = 1 | P3.3<br>TCON.3 |
| Timer 1 Overflow         | TF1                                 | TCON.7         |
| Serial Interface Unit SI |                                     | STS.4          |

External Interrupt control bits IT0 and IT1 are in TCON.0 and TCON.2, respectively. Reset leaves all flags inactive, with IT0 and IT1 cleared.

All the interrupt flags can be set or cleared by software, with the same effect as by hardware.

The Enable and Priority Control Registers are shown below. All of these control bits are set or cleared by software. All are cleared by reset.

### IE: Interrupt Enable Register (bit-addressable)

|      |    |   |   |    |     |     |     |     |
|------|----|---|---|----|-----|-----|-----|-----|
| Bit: | 7  | 6 | 5 | 4  | 3   | 2   | 1   | 0   |
|      | EA | X | X | ES | ET1 | EX1 | ET0 | EX0 |

where:

- EA disables all interrupts. If EA = 0, no interrupt will be acknowledged. If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.



- **ES** enables or disables the Serial Interface Unit interrupt. If ES = 0, the Serial Interface Unit interrupt is disabled.
- **ET1** enables or disables the Timer 1 Overflow interrupt. If ET1 = 0, the Timer 1 interrupt is disabled.
- **EX1** enables or disables External Interrupt 1. If EX1 = 0, External Interrupt 1 is disabled.
- **ET0** enables or disables the Timer 0 Overflow interrupt. If ET0 = 0, the Timer 0 interrupt is disabled.

#### IP: Interrupt Priority Register (bit-addressable)

|      |   |   |   |    |     |     |     |     |
|------|---|---|---|----|-----|-----|-----|-----|
| Bit: | 7 | 6 | 5 | 4  | 3   | 2   | 1   | 0   |
|      | X | X | X | PS | PT1 | PX1 | PT0 | PX0 |

where:

- **PS** defines the Serial Interface Unit interrupt priority level. PS = 1 programs it to the higher priority level.
- **PT1** defines the Timer 1 interrupt priority level. PT1 = 1 programs it to the higher priority level.
- **PX1** defines the External Interrupt 1 priority level. PX1 = 1 programs it to the higher priority level.
- **PT0** defines the Timer 0 interrupt priority level. PT0 = 1 programs it to the higher priority level.
- **PX0** defines the External Interrupt 0 priority level. PX0 = 1 programs it to the higher priority level.

### 17.2 Memory Organization Details

In the 8044 family the memory is organized over three address spaces and the program counter. The memory spaces shown in Figure 18-2 are the:

- 64K-byte Program Memory address space
- 64K-byte External Data Memory address space
- 320-byte Internal Data Memory address space

The 16-bit Program Counter register provides the 8044 with its 64K addressing capabilities. The Program Counter allows the user to execute calls and branches to

any location within the Program Memory space. There are no instructions that permit program execution to move from the Program Memory space to any of the data memory spaces.

In the 8044 and 8744 the lower 4K of the 64K Program Memory address space is filled by internal ROM and EPROM, respectively. By tying the EA pin high, the processor can be forced to fetch from the internal ROM/EPROM for Program Memory addresses 0 through 4K. Bus expansion for accessing Program Memory beyond 4K is automatic since external instruction fetches occur automatically when the Program Counter increases above 4095. If the EA pin is tied low all Program Memory fetches are from external memory. The execution speed of the 8044 is the same regardless of whether fetches are from internal or external Program Memory. If all program storage is on-chip, byte location 4095 should be left vacant to prevent an undesired prefetch from external Program Memory address 4096.

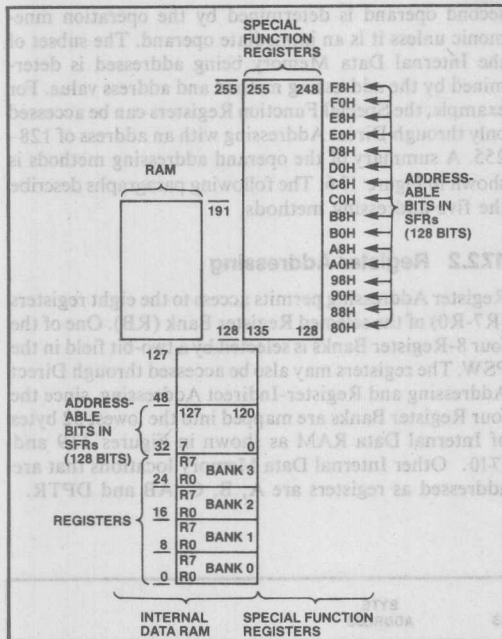
Certain locations in Program Memory are reserved for specific programs. Locations 0000 through 0002 are reserved for the initialization program. Following reset, the CPU always begins execution at location 0000. Locations 0003 through 0042 are reserved for the five interrupt-request service programs. Each resource that can request an interrupt requires that its service program be stored at its reserved location.

The 64K-byte External Data Memory address space is automatically accessed when the MOVX instruction is executed.

Functionally the Internal Data Memory is the most flexible of the address spaces. The Internal Data Memory space is subdivided into a 256-byte Internal Data RAM address space and a 128-byte Special Function Register address space as shown in Figure 17-3.

The Internal Data RAM address space is 0 to 255. Four 8-Register Banks occupy locations 0 through 31. The stack can be located anywhere in the Internal Data RAM address space. In addition, 128 bit locations of the on-chip RAM are accessible through Direct Addressing. These bits reside in Internal Data RAM at byte locations 32 through 47. Currently locations 0 through 191 of the Internal Data RAM address space are filled with on-chip RAM.

The stack depth is limited only by the available Internal Data RAM, thanks to an 8-bit reloadable Stack Pointer. The stack is used for storing the Program Counter during subroutine calls and may be used for passing parameters. Any byte of Internal Data RAM or Special Function Register accessible through Direct Addressing can be pushed/popped.



**Figure 17-3. Internal Data Memory Address Space**

The Special Function Register address space is 128 to 255. All registers except the Program Counter and the four 8-Register Banks reside here. Memory mapping the Special Function Registers allows them to be accessed as easily as internal RAM. As such, they can be operated on by most instructions. In the overlapping memory space (address 128-191), indirect addressing is used to access RAM, and direct addressing is used to access the SFR's. The SFR's at addresses 192-255 are also accessed using direct addressing. The Special Function Registers are listed in Figure 17-4. Their mapping in the Special Function Register address space is shown in Figures 17-5 and 17-6.

Performing a read from a location of the Internal Data memory where neither a byte of Internal Data RAM (i.e., RAM addresses 192-255) nor a Special Function Register exists will access data of indeterminable value.

Architecturally, each memory space is a linear sequence of 8-bit wide bytes. By Intel convention the storage of multi-byte address and data operands in program and data memories is the least significant byte at the low-order address and the most significant byte at the high-order address. Within byte X, the most significant bit is

**ARITHMETIC REGISTERS:**  
Accumulator\*, B register\*,  
Program Status Word\*

**POINTERS:**  
Stack Pointer, Data Pointer (high & low)

**PARALLEL I/O PORTS:**  
Port 3\*, Port 2\*, Port 1\*, Port 0\*

**INTERRUPT SYSTEM:**  
Interrupt Priority Control\*,  
Interrupt Enable Control\*

**TIMERS:**  
Timer MODE, Timer CONTROL\*, Timer 1 (high & low), Timer 0 (high & low)

**SERIAL INTERFACE UNIT:**  
Transmit Buffer Start,  
Transmit Buffer Length,  
Transmit Control Byte,  
Send Count Receive Count\*,  
DMA Count,  
Station Address  
Receive Field Length  
Receive Buffer Start  
Receive Buffer Length  
Receive Control Byte,  
Serial Mode,  
Status Register.\*

\* Bits in these registers are bit addressable.

**Figure 17-4. Special Function Registers**

represented by X.7 while the least significant bit is X.0. Any deviation from these conventions will be explicitly stated in the text.

### 17.2.1 Operand Addressing

There are five methods of addressing source operands. They are Register Addressing, Direct Addressing, Register-Indirect Addressing, Immediate Addressing and Base-Register-plus Index-Register-Indirect Addressing. The first three of these methods can also be used to address a destination operand. Since operations in the 8044 require 0 (NOP only), 1, 2, 3 or 4 operands, these five addressing methods are used in combinations to provide the 8044 with its 21 addressing modes.

Most instructions have a "destination, source" field that specifies the data type, addressing methods and operands involved. For operations other than moves, the destination operand is also a source operand. For example, in "subtract-with-borrow A, #5" the A register receives the result of the value in register A minus 5, minus C.

Most operations involve operands that are located in Internal Data Memory. The selection of the Program Memory space or External Data Memory space for a

ARITHMETIC REGISTERS:  
Accumulator\*, B register\*,  
Program Status Word\*

POINTERS:  
Stack Pointer, Data Pointer (high &  
low)

PARALLEL I/O PORTS:  
Port 3\*, Port 2\*, Port 1\*, Port 0\*

INTERRUPT SYSTEM:  
Interrupt Priority Control\*,  
Interrupt Enable Control\*

TIMERS:  
Timer Mode, Timer Control\*, Timer 1  
(high & low), Timer 0 (high & low)

SERIAL INTERFACE UNIT:  
Serial Mode, Status/Command\*,  
Send/Receive Count\*, Station Address,  
Transmit Buffer Start Address,  
Transmit Buffer Length,  
Transmit Control Byte,  
Receive Buffer Start Address,  
Receive Buffer Length,  
Receive Field Length,  
Receive Control Byte,  
DMA Count,  
FIFO (three bytes),  
SIU Controller State Counter

\* Bits in these registers are bit-addressable

Figure 17-5. Mapping of Special Function Registers

second operand is determined by the operation mnemonic unless it is an immediate operand. The subset of the Internal Data Memory being addressed is determined by the addressing method and address value. For example, the Special Function Registers can be accessed only through Direct Addressing with an address of 128–255. A summary of the operand addressing methods is shown in Figure 17-6. The following paragraphs describe the five addressing methods.

## 17.2.2 Register Addressing

Register Addressing permits access to the eight registers (R7-R0) of the selected Register Bank (RB). One of the four 8-Register Banks is selected by a two-bit field in the PSW. The registers may also be accessed through Direct Addressing and Register-Indirect Addressing, since the four Register Banks are mapped into the lowest 32 bytes of Internal Data RAM as shown in Figures 17-9 and 17-10. Other Internal Data Memory locations that are addressed as registers are A, B, C, AB and DPTR.

| REGISTER NAMES             | SYMBOLIC ADDRESS | BIT ADDRESS     | BYTE ADDRESS |
|----------------------------|------------------|-----------------|--------------|
| B REGISTER                 | B                | 247 through 240 | 240 (F0H) ←  |
| ACCUMULATOR                | ACC              | 231 through 224 | 224 (E0H) ←  |
| *THREE BYTE FIFO           | FIFO             |                 | 223 (DFH) ←  |
|                            | FIFO             |                 | 222 (DEH) ←  |
|                            | FIFO             |                 | 221 (DDH) ←  |
| TRANSMIT BUFFER START      | TBS              |                 | 220 (DCH) ←  |
| TRANSMIT BUFFER LENGTH     | TBL              |                 | 219 (DBH) ←  |
| TRANSMIT CONTROL BYTE      | TCB              |                 | 218 (DAH) ←  |
| *SIU STATE COUNTER         | SIUST            |                 | 217 (D9H) ←  |
| SEND COUNT RECEIVE COUNT   | NSNR             | 223 through 216 | 216 (D8H) ←  |
| PROGRAM STATUS WORD        | PSW              | 215 through 208 | 208 (D0H) ←  |
| *DMA COUNT                 | DMA CNT          |                 | 207 (CFH) ←  |
| STATION ADDRESS            | STAD             |                 | 206 (CEH) ←  |
| RECEIVE FIELD LENGTH       | RFL              |                 | 205 (CDH) ←  |
| RECEIVE BUFFER START       | RBS              |                 | 204 (CCH) ←  |
| RECEIVE BUFFER LENGTH      | RBL              |                 | 203 (CBH) ←  |
| RECEIVE CONTROL BYTE       | RCB              |                 | 202 (CAH) ←  |
| SERIAL MODE                | SMD              |                 | 201 (C9H) ←  |
| STATUS REGISTER            | STS              | 207 through 200 | 200 (C8H) ←  |
| INTERRUPT PRIORITY CONTROL | IP               | 191 through 184 | 184 (B8H) ←  |
| PORT 3                     | P3               | 183 through 176 | 176 (B0H) ←  |
| INTERRUPT ENABLE CONTROL   | IE               | 175 through 168 | 168 (A8H) ←  |
| PORT 2                     | P2               | 167 through 160 | 160 (A0H) ←  |
| PORT 1                     | P1               | 151 through 144 | 144 (90H) ←  |
| TIMER HIGH 1               | TH1              |                 | 141 (8DH) ←  |
| TIMER HIGH 0               | TH0              |                 | 140 (8CH) ←  |
| TIMER LOW 1                | TL1              |                 | 139 (8BH) ←  |
| TIMER LOW 0                | TL0              |                 | 138 (8AH) ←  |
| TIMER MODE                 | TMOD             |                 | 137 (89H) ←  |
| TIMER CONTROL              | TCON             | 143 through 136 | 136 (88H) ←  |
| DATA POINTER HIGH          | DPH              |                 | 131 (83H) ←  |
| DATA POINTER LOW           | DPL              |                 | 130 (82H) ←  |
| STACK POINTER              | SP               |                 | 129 (81H) ←  |
| PORT 0                     | P0               | 135 through 128 | 128 (80H) ←  |

\*ICE Support Hardware registers. Under normal operating conditions there is no need for the CPU to access these registers.

Figure 17-6. Mapping of Special Function Registers

| Direct<br>Byte<br>Address | Bit Address<br>(MSB)            | (LSB) | Hardware<br>Register<br>Symbol |
|---------------------------|---------------------------------|-------|--------------------------------|
| 240                       | F7 F6 F5 F4 F3 F2 F1 F0         |       | 8                              |
| 224                       | E7 E6 E5 E4 E3 E2 E1 E0         |       | ACC                            |
| 216                       | NS2 NS1 NS0 SES NR2 NR1 NR0 SER |       | NSNR                           |
| 204                       | DF DE DD DC DB DA D9 D8         |       | PSW                            |
| 200                       | CY AC FO RS1 RSO OV P           |       | STS                            |
| 184                       | D7 D6 D5 D4 D3 D2 D1 D0         |       | 1P                             |
| 176                       | TBF RE RTS SI BV CPB AM RBP     |       | P3                             |
| 168                       | CF CE CD CC CB CA C9 C8         |       | 1E                             |
| 160                       | PS PT1 PX1 PT0 PX0              |       | P2                             |
| 144                       | B7 B6 B5 B4 B3 B2 B1 B0         |       | P1                             |
| 136                       | EA E5 ET1 EX1 ET0 EX0           |       | TCN                            |
| 128                       | AF — — AC AB AA A9 A8           |       | P0                             |
|                           | A7 A6 A5 A4 A3 A2 A1 A0         |       |                                |
|                           | TF1 TR1 TF0 TR0 IE1 IT1 IE0 IT0 |       |                                |
|                           | 8F 8E 8D 8C 8B 8A 89 88         |       |                                |
|                           | 87 86 85 84 83 82 81 80         |       |                                |

Figure 17-7. Special Function Register Bit Address

| RAM<br>BYTE<br>ADDRESS | Bit Address<br>(MSB)    | (LSB) |
|------------------------|-------------------------|-------|
| 2FH                    | 7E 7D 7C 7B 7A 79 78 77 | 47    |
| 2EH                    | 76 75 74 73 72 71 70 6F | 46    |
| 2DH                    | 6E 6D 6C 6B 6A 69 68 67 | 45    |
| 2CH                    | 66 65 64 63 62 61 60 5F | 44    |
| 2BH                    | 5E 5D 5C 5B 5A 59 58 57 | 43    |
| 2AH                    | 56 55 54 53 52 51 50 4F | 42    |
| 29H                    | 4E 4D 4C 4B 4A 49 48 47 | 41    |
| 28H                    | 46 45 44 43 42 41 40 3F | 40    |
| 27H                    | 3E 3D 3C 3B 3A 39 38 37 | 39    |
| 26H                    | 36 35 34 33 32 31 30 2F | 38    |
| 25H                    | 2E 2D 2C 2B 2A 29 28 27 | 37    |
| 24H                    | 26 25 24 23 22 21 20 1F | 36    |
| 23H                    | 1E 1D 1C 1B 1A 19 18 17 | 35    |
| 22H                    | 16 15 14 13 12 11 10 0F | 34    |
| 21H                    | 0E 0D 0C 0B 0A 09 08 07 | 33    |
| 20H                    | 06 05 04 03 02 01 00    | 32    |
| 1FH                    |                         | 31    |
| 1EH                    |                         | 30    |
| 1DH                    |                         | 29    |
| 1CH                    |                         | 28    |
| 1BH                    |                         | 27    |
| 1AH                    |                         | 26    |
| 19H                    |                         | 25    |
| 18H                    |                         | 24    |
| 17H                    |                         | 23    |
| 16H                    |                         | 22    |
| 15H                    |                         | 21    |
| 14H                    |                         | 20    |
| 13H                    |                         | 19    |
| 12H                    |                         | 18    |
| 11H                    |                         | 17    |
| 10H                    |                         | 16    |
| 0FH                    |                         | 15    |
| 0EH                    |                         | 14    |
| 0DH                    |                         | 13    |
| 0CH                    |                         | 12    |
| 0BH                    |                         | 11    |
| 0AH                    |                         | 10    |
| 09H                    |                         | 9     |
| 08H                    |                         | 8     |
| 07H                    |                         | 7     |
| 06H                    |                         | 6     |
| 05H                    |                         | 5     |
| 04H                    |                         | 4     |
| 03H                    |                         | 3     |
| 02H                    |                         | 2     |
| 01H                    |                         | 1     |
| 00H                    |                         | 0     |

Figure 17-9. RAM Bit Addresses

|  |
|--|
| <ul style="list-style-type: none"> <li>Register Addressing <ul style="list-style-type: none"> <li>R7-R0</li> <li>A,B,C (bit), AB (two bytes), DPTR (double byte)</li> </ul> </li> <li>Direct Addressing <ul style="list-style-type: none"> <li>Lower 128 bytes of Internal Data RAM</li> <li>Special Function Registers</li> <li>128 bits in subset of Special Function Register address space</li> </ul> </li> <li>Register-Indirect Addressing <ul style="list-style-type: none"> <li>Internal Data RAM [<math>@R1</math>, <math>@R0</math>, <math>@SP</math> (PUSH and POP only)]</li> <li>Least Significant Nibbles in Internal Data RAM (<math>@R1</math>, <math>@R0</math>)</li> <li>External Data Memory (<math>@R1</math>, <math>@R0</math>, <math>@DPTR</math>)</li> </ul> </li> <li>Immediate Addressing <ul style="list-style-type: none"> <li>Program Memory (in-code constant)</li> </ul> </li> <li>Base-Register-plus Index-Register-Indirect Addressing <ul style="list-style-type: none"> <li>Program Memory (<math>@DPTR + A</math>, <math>@PC + A</math>)</li> </ul> </li> </ul> |
|--|

Figure 17-8. Operand Addressing Methods

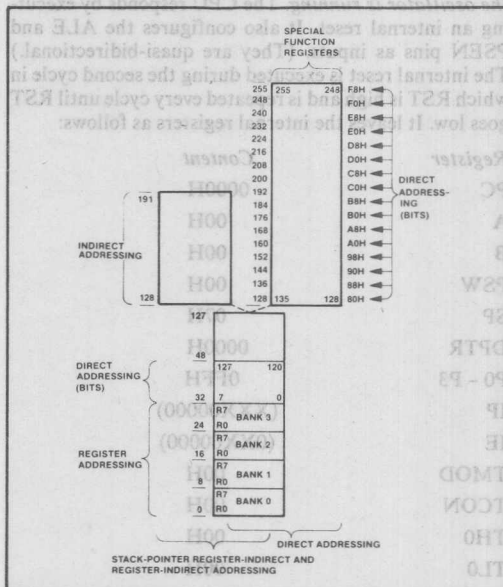


Figure 17-10. Addressing Operands in Internal Data Memory



### 17.2.3 Direct Addressing

Direct Addressing provides the only means of accessing the memory-mapped byte-wide Special Function Registers and memory mapped bits within the Special Function Registers and Internal Data RAM. Direct Addressing of bytes may also be used to access the lower 128 bytes of Internal Data RAM. Direct Addressing of bits gains access to a 128 bit subset of the Internal Data RAM and 128 bit subset of the Special Function Registers as shown in Figures 17-5, 17-6, 17-9, and 17-10.

Register-Indirect Addressing using the content of R1 or R0 in the selected Register Bank, or using the content of the Stack Pointer (PUSH and POP only), addresses the Internal Data RAM. Register-Indirect Addressing is also used for accessing the External Data Memory. In this case, either R1 or R0 in the selected Register Bank may be used for accessing locations within a 256-byte block. The block number can be preselected by the contents of a port. The 16-bit Data Pointer may be used for accessing any location within the full 64K external address space.

### 17.3 RESET

Reset is accomplished by holding the RST pin high for at least two machine cycles (24 oscillator periods) *while the oscillator is running*. The CPU responds by executing an internal reset. It also configures the ALE and PSEN pins as inputs. (They are quasi-bidirectional.) The internal reset is executed during the second cycle in which RST is high and is repeated every cycle until RST goes low. It leaves the internal registers as follows:

| Register | Content    |
|----------|------------|
| PC       | 0000H      |
| A        | 00H        |
| B        | 00H        |
| PSW      | 00H        |
| SP       | 07H        |
| DPTR     | 0000H      |
| P0 - P3  | 0FFH       |
| IP       | (XXX00000) |
| IE       | (0XX00000) |
| TMOD     | 00H        |
| TCON     | 00H        |
| TH0      | 00H        |
| TL0      | 00H        |
| TH1      | 00H        |
| TL1      | 00H        |

|         |            |
|---------|------------|
| SMD     | 00H        |
| STS     | 00H        |
| NSNR    | 00H        |
| STAD    | 00H        |
| TBS     | 00H        |
| TBL     | 00H        |
| TCB     | 00H        |
| RBS     | 00H        |
| RBL     | 00H        |
| RFL     | 00H        |
| RCB     | 00H        |
| DMA CNT | 00H        |
| FIFO1   | 00H        |
| FIFO2   | 00H        |
| FIFO3   | 00H        |
| SIUST   | 01H        |
| PCON    | (0XXXXXXX) |

The internal RAM is not affected by reset. When VCC is turned on, the RAM content is indeterminate unless VPD was applied prior to VCC being turned off (see Power Down Operation.)

### 17.4 RUP1™-44 FAMILY PIN DESCRIPTION

**VSS:** Circuit ground potential.

**VCC:** Supply voltage during programming (of the 8744), verification (of the 8044 or 8744), and normal operation.

**Port 0:** Port 0 is an 8-bit open drain bidirectional I/O port. It is also the multiplexed low-order address and data bus during accesses to external memory (during which accesses it activates internal pullups). It also outputs instruction bytes during program verification. (External pullups are required during program verification.) Port 0 can sink eight LS TTL inputs.

**Port 1:** Port 1 is an 8-bit bidirectional I/O port with internal pullups. It receives the low-order address byte during program verification in the 8044 or 8744. Port 1 can sink/source four LS TTL inputs. It can drive MOS inputs without external pullups.

Two of the Port 1 pins serve alternate functions, as listed below:

| Port Pin | Alternate Function   |
|----------|--|
| P1.6     | RTS (Request to Send). In a non-loop configuration, RTS signals that the 8044 is ready to transmit data. |

P1.7  $\overline{\text{CTS}}$  (Clear to Send). In a non-loop configuration,  $\overline{\text{CTS}}$  signals to the 8044 that the receiving station is ready to accept data.

**Port 2:** Port 2 is an 8-bit bidirectional I/O port with internal pullups. It emits the high-order address byte during accesses to external memory. It also receives the high-order address bits and control signals during program verification in the 8044 or 8744. Port 2 can sink/source four LS TTL inputs. It can drive MOS inputs without external pullups.

**Port 3:** Port 3 is an 8-bit bidirectional I/O port with internal pullups. Port 3 can sink/source four LS TTL inputs. It can drive MOS inputs without external pullups.

Port 3 pins also serve alternate functions, as listed below:

| Port Pin | Alternate Function   |
|----------|--|
| P3.0     | RXD (serial input port in loop configuration). I/O (data direction control in non-loop configuration). |
| P3.1     | TXD (serial output port in loop configuration). DATA input/output pin in non-loop configuration.       |
| P3.2     | $\overline{\text{INT0}}$ (external interrupt)  |
| P3.3     | $\overline{\text{INT1}}$ (external interrupt)  |
| P3.4     | T0 (Timer 0 external input)  |
| P3.5     | T1 (Timer 1 external input) SCLK (Serial Data Clock Input)   |
| P3.6     | $\overline{\text{WR}}$ (external Data Memory write strobe)   |
| P3.7     | $\overline{\text{RD}}$ (external Data Memory read strobe)  |

**RST/VPD:** A high level on this pin for two machine cycles while the oscillator is running resets the device. An

internal pulldown permits Power-On reset using only a capacitor connected to VCC.

**ALE/PROG:** Address Latch Enable output for latching the low byte of the address during accesses to external memory. ALE is activated though for this purpose at a constant rate of 1/6 the oscillator frequency even when external memory is not being accessed. Consequently it can be used for external clocking or timing purposes. (However, one ALE pulse is skipped during each access to external Data Memory.) This pin is also the program pulse input (PROG) during EPROM programming.

**PSEN:** Program Store Enable output is the read strobe to external Program Memory. PSEN is activated twice each machine cycle during fetches from external Program Memory. (However, when executing out of external Program Memory two activations of PSEN are skipped during each access to external Data Memory.) PSEN is not activated during fetches from internal Program Memory.

**$\overline{\text{EA}}$ /VPP:** When EA is held high the CPU executes out of internal Program Memory (unless the Program Counter exceeds 0FFFFH). When  $\overline{\text{EA}}$  is held low the CPU executes only out of external Program Memory. In the 8344,  $\overline{\text{EA}}$  must be externally wired low. In the 8744, this pin also receives the 21V programming supply voltage (VPP) during EPROM programming.

**XTAL1:** Input to the inverting amplifier that forms the oscillator. Should be grounded when an external oscillator is used.

**XTAL2:** Output of the inverting amplifier that forms the oscillator, and input to the internal clock generator. Receives the external oscillator signal when an external oscillator is used.







18

8044 Serial Interface

---

## CHAPTER 18

# THE 8044 SERIAL INTERFACE UNIT

### 18.0 SERIAL INTERFACE

The serial interface provides a high-performance communication link. The protocol used for this communication is based on the IBM Synchronous Data Link Control (SDLC). The serial interface also supports a subset of the ISO HDLC (International Standards Organization High-Level Data Link Control) protocol.

The SDLC/HDLC protocols have been accepted as standard protocols for many high-level teleprocessing systems. The serial interface performs many of the functions required to service the data link without intervention from the 8044's own CPU. The programmer is free to concentrate on the 8044's function as a peripheral controller, rather than having to deal with the details of the communication process.

Five pins on the 8044 are involved with the serial interface (refer to Section 12.4, Family Pin Description, for details):

|        |  |
|--------|--|
| Pin 7  | $\overline{\text{RTS}}/\text{P16}$                   |
| Pin 8  | $\overline{\text{CTS}}/\text{P17}$                   |
| Pin 10 | $\text{I}/\overline{\text{O}}/\text{RXD}/\text{P30}$ |
| Pin 11 | $\text{DATA}/\text{TXD}/\text{P31}$                  |
| Pin 15 | $\text{SCLK}/\text{T1}/\text{P35}$                   |

Figure 18-1 is a functional block diagram of the serial interface unit (SIU). More details on the SIU hardware are given in Section 18.9.

### 18.1 DATA LINK CONFIGURATIONS

The serial interface is capable of operating in three serial data link configurations:

- 1) Half-Duplex, point-to-point
- 2) Half-Duplex, multipoint (with a half-duplex or full-duplex primary)
- 3) Loop

Figure 18-2 shows these three configurations. The RTS (Request to Send) and CTS (Clear to Send) handshaking signals are available in the point-to-point and multipoint configurations.

### 18.2 DATA CLOCKING OPTIONS

The serial interface can operate in an externally clocked mode or in a self clocked mode.

#### Externally Clocked Mode

In the externally clocked mode, a common Serial Data Clock (SCLK on pin 15) synchronizes the serial bit stream. This clock signal may come from the master CPU or primary station, or from an external phase-locked loop local to the 8044. Figure 18-3 illustrates the timing relationships for the serial interface signals when the externally clocked mode is used in point-to-point and multipoint data link configurations.

Incoming data is sampled at the rising edge of SCLK, and outgoing data is shifted out at the falling edge of SCLK. More detailed timing information is given in the 8044 data sheet.

#### Self Clocked (Asynchronous) Mode

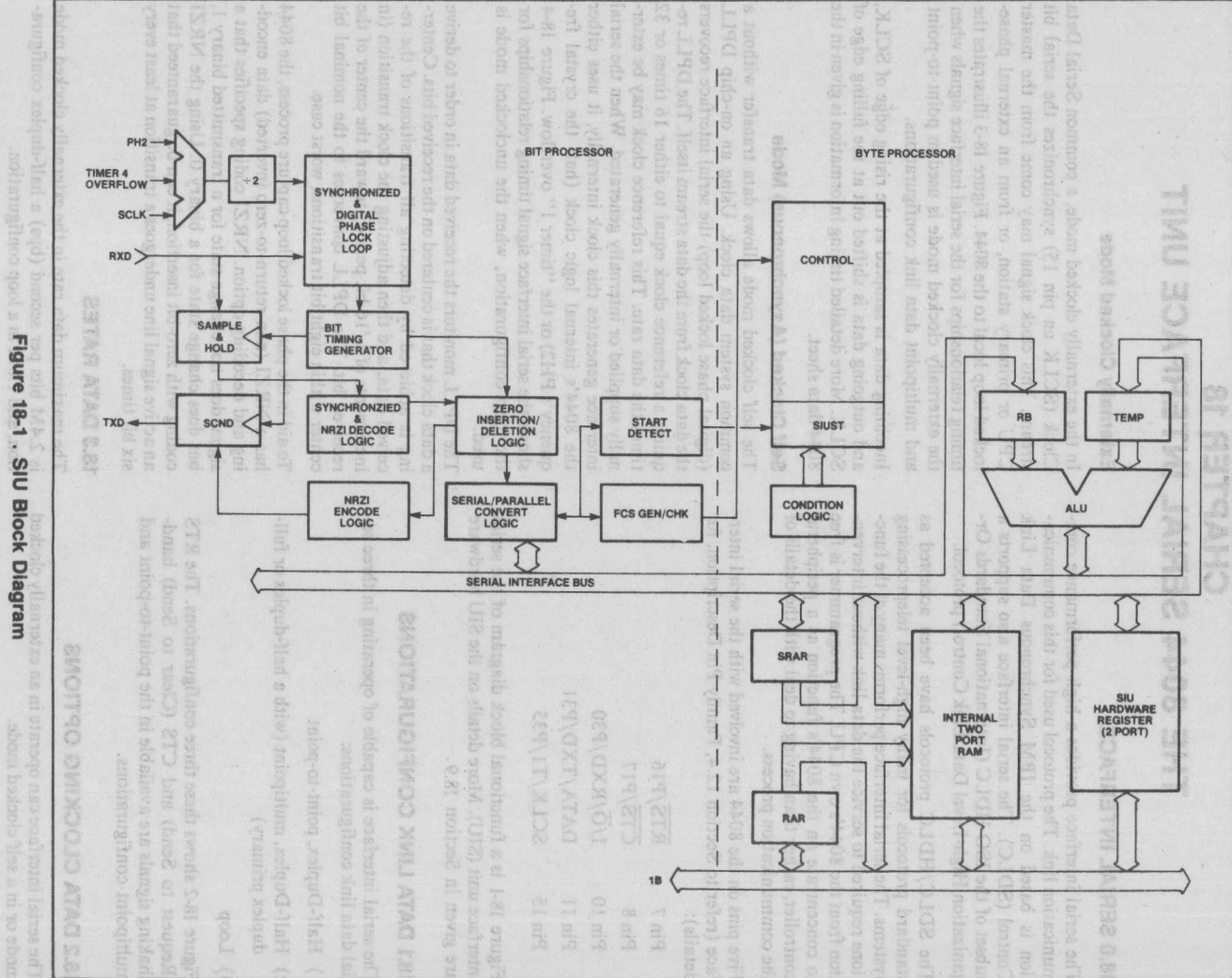
The self clocked mode allows data transfer without a common system data clock. Using an on-chip DPLL (digital phase locked loop) the serial interface recovers the data clock from the data stream itself. The DPLL requires a reference clock equal to either 16 times or 32 times the data rate. This reference clock may be externally supplied or internally generated. When the serial interface generates this clock internally, it uses either the 8044's internal logic clock (half the crystal frequency's PH2) or the "timer 1" overflow. Figure 18-4 shows the serial interface signal timing relationships for the loop configuration, when the unclocked mode is used.

The DPLL monitors the received data in order to derive a data clock that is centered on the received bits. Centering is achieved by detecting all transitions of the received data, and then adjusting the clock transition (in increments of 1/16 bit period) toward the center of the received bit. The DPLL converges to the nominal bit center within eight bit transitions, worst case.

To aid in the phase locked loop capture process, the 8044 has a NRZI (non-return-to-zero inverted) data encoding and decoding option. NRZI coding specifies that a signal does not change state for a transmitted binary 1, but does change state for a binary 0. Using the NRZI coding with zero-bit insertion, it can be guaranteed that an active signal line undergoes a transition at least every six bit times.

### 18.3 DATA RATES

The maximum data rate in the externally clocked mode is 2.4M bits per second (bps) a half-duplex configuration, and 1.0M in a loop configuration.



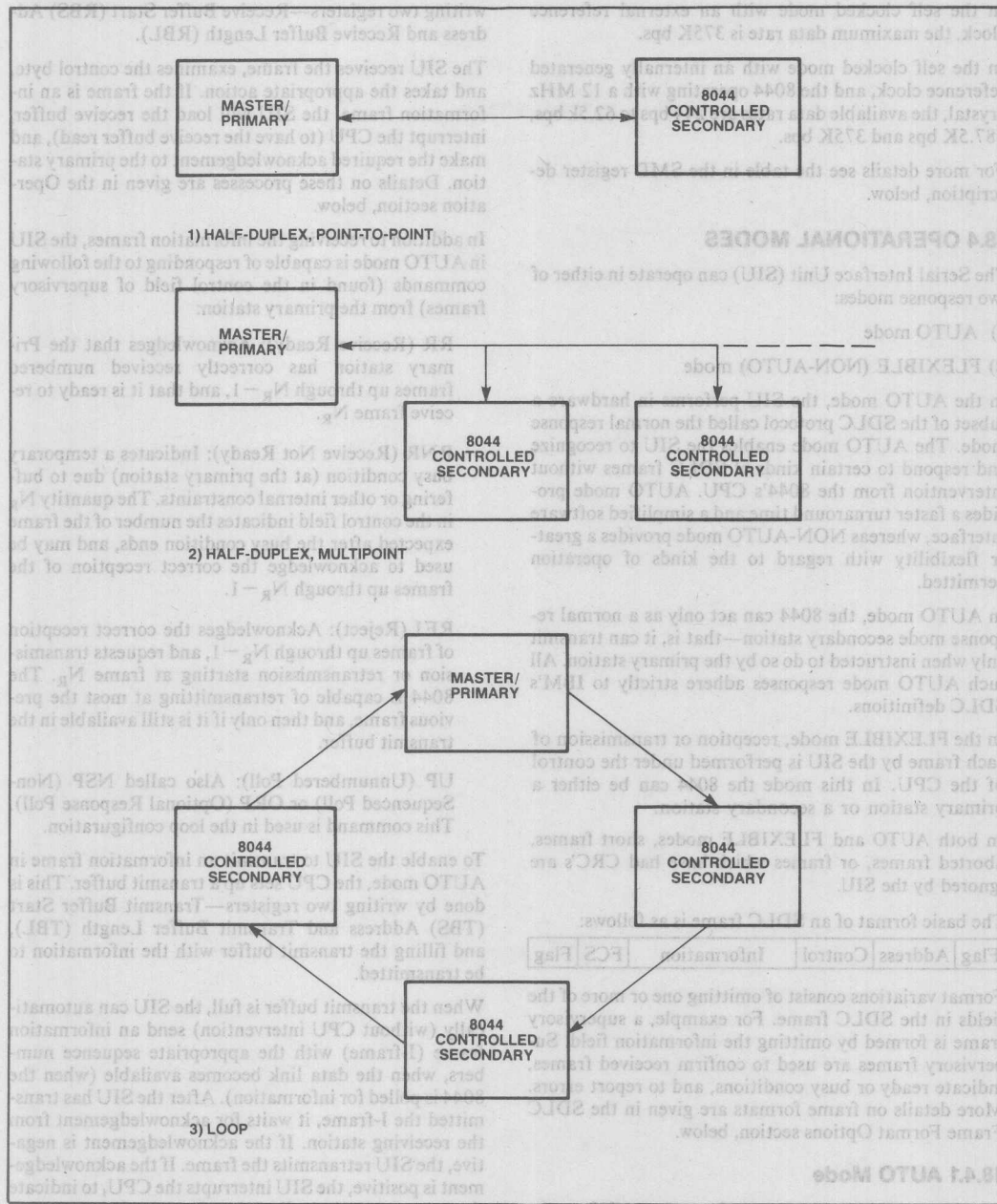


Figure 18-2 . RUPI-44 Data Link Configurations



In the self clocked mode with an external reference clock, the maximum data rate is 375K bps.

In the self clocked mode with an internally generated reference clock, and the 8044 operating with a 12 MHz crystal, the available data rates are 244 bps to 62.5k bps, 187.5K bps and 375K bps.

For more details see the table in the SMD register description, below.

## 18.4 OPERATIONAL MODES

The Serial Interface Unit (SIU) can operate in either of two response modes:

- 1) AUTO mode
- 2) FLEXIBLE (NON-AUTO) mode

In the AUTO mode, the SIU performs in hardware a subset of the SDLC protocol called the normal response mode. The AUTO mode enables the SIU to recognize and respond to certain kinds of SDLC frames without intervention from the 8044's CPU. AUTO mode provides a faster turnaround time and a simplified software interface, whereas NON-AUTO mode provides a greater flexibility with regard to the kinds of operation permitted.

In AUTO mode, the 8044 can act only as a normal response mode secondary station—that is, it can transmit only when instructed to do so by the primary station. All such AUTO mode responses adhere strictly to IBM's SDLC definitions.

In the FLEXIBLE mode, reception or transmission of each frame by the SIU is performed under the control of the CPU. In this mode the 8044 can be either a primary station or a secondary station.

In both AUTO and FLEXIBLE modes, short frames, aborted frames, or frames which have had CRC's are ignored by the SIU.

The basic format of an SDLC frame is as follows:

| Flag | Address | Control | Information | FCS | Flag |
|------|---------|---------|-------------|-----|------|
|------|---------|---------|-------------|-----|------|

Format variations consist of omitting one or more of the fields in the SDLC frame. For example, a supervisory frame is formed by omitting the information field. Supervisory frames are used to confirm received frames, indicate ready or busy conditions, and to report errors. More details on frame formats are given in the SDLC Frame Format Options section, below.

### 18.4.1 AUTO Mode

To enable the SIU to receive a frame in AUTO mode, the 8044 CPU sets up a receive buffer. This is done by

writing two registers—Receive Buffer Start (RBS) Address and Receive Buffer Length (RBL).

The SIU receives the frame, examines the control byte, and takes the appropriate action. If the frame is an information frame, the SIU will load the receive buffer, interrupt the CPU (to have the receive buffer read), and make the required acknowledgement to the primary station. Details on these processes are given in the Operation section, below.

In addition to receiving the information frames, the SIU in AUTO mode is capable of responding to the following commands (found in the control field of supervisory frames) from the primary station:

**RR (Receive Ready):** Acknowledges that the Primary station has correctly received numbered frames up through  $N_R - 1$ , and that it is ready to receive frame  $N_R$ .

**RNR (Receive Not Ready):** Indicates a temporary busy condition (at the primary station) due to buffering or other internal constraints. The quantity  $N_R$  in the control field indicates the number of the frame expected after the busy condition ends, and may be used to acknowledge the correct reception of the frames up through  $N_R - 1$ .

**REJ (Reject):** Acknowledges the correct reception of frames up through  $N_R - 1$ , and requests transmission or retransmission starting at frame  $N_R$ . The 8044 is capable of retransmitting at most the previous frame, and then only if it is still available in the transmit buffer.

**UP (Unnumbered Poll):** Also called NSP (Non-Sequenced Poll) or ORP (Optional Response Poll). This command is used in the loop configuration.

To enable the SIU to transmit an information frame in AUTO mode, the CPU sets up a transmit buffer. This is done by writing two registers—Transmit Buffer Start (TBS) Address and Transmit Buffer Length (TBL), and filling the transmit buffer with the information to be transmitted.

When the transmit buffer is full, the SIU can automatically (without CPU intervention) send an information frame (I-frame) with the appropriate sequence numbers, when the data link becomes available (when the 8044 is polled for information). After the SIU has transmitted the I-frame, it waits for acknowledgement from the receiving station. If the acknowledgement is negative, the SIU retransmits the frame. If the acknowledgement is positive, the SIU interrupts the CPU, to indicate that the transmit buffer may be reloaded with new information.

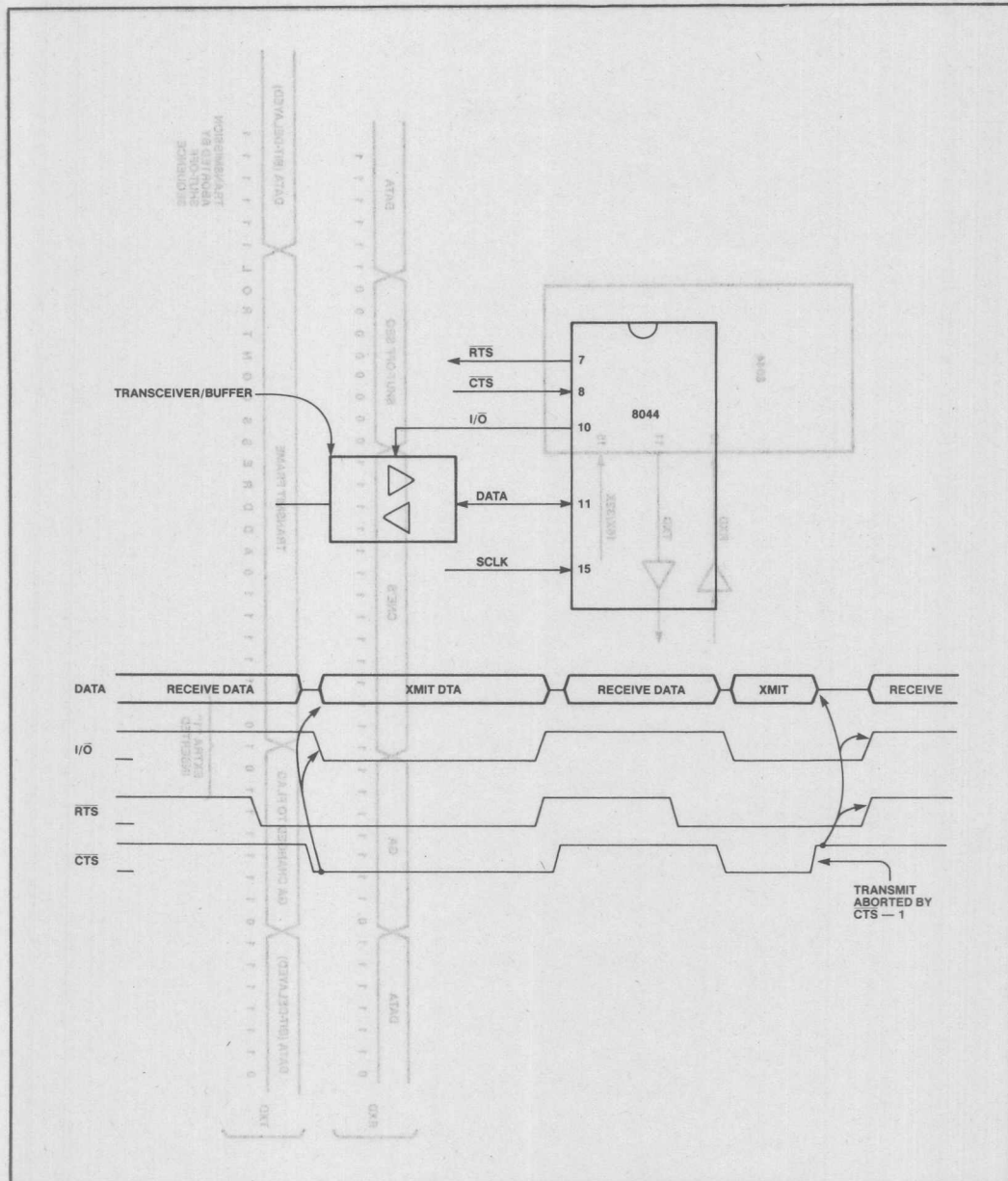


Figure 18-3. Serial Interface Timing—Clocked Mode



In addition to transmitting the information frames, the SIU in AUTO mode is capable of sending the following responses to the primary station:

**RR (Receive Ready):** Acknowledges that the 8044 has correctly received numbered frames up through  $N_R - 1$ , and that it is ready to receive frame  $N_R$ .

**RNR (Receive Not Ready):** Indicates a temporary busy condition (at the 8044) due to buffering or other internal constraints. The quantity  $N_R$  in the control field indicates the number of the frame expected after the busy condition ends, and acknowledges the correct reception of the frames up through  $N_R - 1$ .

#### 18.4.2 FLEXIBLE Mode

In the FLEXIBLE (or non-auto) mode, all reception and transmission is under the control of the CPU. The full SDLC and HDLC protocols can be implemented, as well as any bit-synchronous variants of these protocols.

FLEXIBLE mode provides more flexibility than AUTO mode, but it requires more CPU overhead, and much longer recognition and response times. This is especially true when the CPU is servicing an interrupt that has higher priority than the interrupts from the SIU.

In FLEXIBLE mode, when the SIU receives a frame, it interrupts the CPU. The CPU then reads the control byte from the Receive Control Byte (RCB) register. If the received frame is an information frame, the CPU also reads the information from the receive buffer, according to the values in the Receive Buffer Start (RBS) address register and the Received Field Length (RFL) register.

In FLEXIBLE mode, the 8044 can initiate transmissions without being polled, and thus it can act as the primary station. To initiate transmission or to generate a response, the CPU sets up and enables the SIU. The SIU then formats and transmits the desired frame. Upon completion of the transmission, without waiting for a positive acknowledgement from the receiving station, the SIU interrupts the CPU.

#### 18.5 8044 FRAME FORMAT OPTIONS

As mentioned above, variations on the basic SDLC frame consist of omitting one or more of the fields. The choice of which fields to omit, as well as the selection of AUTO mode versus FLEXIBLE mode, is specified by the settings of the following three bits in the Serial Mode Register (SMD) and the Status/Control Register (STS):

**SMD Bit 0: NFCS (No Frame Check Sequence)**

**SMD Bit 1: NB (Non-Buffered Mode—No Control Field)**

**STS Bit 1: AM (AUTO Mode or Addressed Mode)**

Figure 18-5 shows how these three bits control the frame format.

The following paragraphs discuss some properties of the standard SDLC format, and the significance of omitting some of the fields.

#### 18.5.1 Standard SDLC Format

The standard SDLC format consists of an opening flag, an 8-bit address field, and 8-bit control field, an n-byte information field, a 16-bit Frame Check Sequence (FCS), and a closing flag. The FCS is based on the CCITT-CRC polynomial ( $X^{16} + X^{12} + X^5 + 1$ ). The address and control fields may not be extended. Within the 8044, the address field is held in the Station Address (STAD) register, and the control field is held in the Receive Control Byte (RCB) or Transmit Control Byte (TCB) register. The standard SDLC format may be used in either AUTO mode or FLEXIBLE mode.

#### 18.5.2 No Control Field (Non-Buffered Mode)

When the control field is not present, the RCB and TCB registers are not used. The information field begins immediately after the address field, or, if the address field is also absent, immediately after the opening flag. The entire information field is stored in the 8044's on-chip RAM. If there is no control field, FLEXIBLE mode must be used. Control information may, of course, be present in the information field, and in this manner the No Control Field option may be used for implementing extended control fields.

#### 18.5.3 No Control Field and No Address Field

The No Address Field option is available only in conjunction with the No Control Field option. The STAD, RCB, and TCB registers are not used. When both these fields are absent, the information field begins immediately after the opening flag. The entire information field is stored in on-chip RAM. FLEXIBLE mode must be used. Formats without an address field have the following applications:

Point-to-point data links (where no addressing is necessary)

Monitoring line activity (receiving all messages regardless of the address field)

Extended addressing



| FRAME OPTION  | NFCS | NB | AM | FRAME FORMAT  |
|---|------|----|----|---------------|
| Standard SDLC<br>FLEXIBLE Mode  | 0    | 0  | 0  | F A C I FCS F |
| Standard SDLC<br>AUTO Mode  | 0    | 0  | 1  | F A C I FCS F |
| No Control Field<br>FLEXIBLE Mode                                     | 0    | 1  | 1  | F A I FCS F   |
| No Control Field<br>No Address Field<br>FLEXIBLE Mode                 | 0    | 1  | 0  | F I FCS F     |
| No FCS Field<br>FLEXIBLE Mode   | 1    | 0  | 0  | F A C I F     |
| No FCS Field<br>AUTO Mode   | 1    | 0  | 1  | F A C I F     |
| No FCS Field<br>No Control Field<br>FLEXIBLE Mode                     | 1    | 1  | 1  | F A I F       |
| No FCS Field<br>No Control Field<br>No Address Field<br>FLEXIBLE Mode | 1    | 1  | 0  | F I F         |

**Key to Abbreviations:**  
 F = Flag (01111110)      I = Information Field  
 A = Address Field      FCS = Frame Check Sequence  
 C = Control Field  
 Note: The AM bit is AUTO mode control bit when NB = 0, and Address Mode control bit when NB = 1.

Figure 18-5 . Frame Format Options

#### 18.5.4 No FCS Field

In the normal case (NFCS=0), the last 16 bits before the closing flag are the Frame Check Sequence (FCS) field. These bits are not stored in the 8044's RAM. Rather, they are used to compute a cyclic redundancy check (CRC) on the data in the rest of the frame. A received frame with a CRC error (incorrect FCS) is ignored. In transmission, the FCS field is automatically computed by the SIU, and placed in the transmitted frame just prior to the closing flag.

The NFCS bit (SMD Bit 0) gives the user the capability of overriding this automatic feature. When this bit is set (NFCS=1), all bits from the beginning of the information field to the beginning of the closing flag are treated as part of the information field, and are stored in the on-chip RAM. No FCS checking is done on the received frames, and no FCS is generated for the transmitted frames. The No FCS Field option may be used in conjunction with any of the other options. It is typically used in FLEXIBLE mode, although it does not strictly include AUTO mode. Use of the No FCS Field option

AUTO Mode may, however, result in SDLC protocol violations, since the data integrity is not checked by the SIU.

Formats without an FCS field have the following applications:

Receiving and transmitting frames without verifying data integrity

Using an alternate data verification algorithm

Using an alternate CRC-16 polynomial (such as  $X^{16} + X^{15} + X^2 + 1$ ), or a 32-bit CRC

Performing data link diagnosis by forcing false CRCs to test error detection mechanisms

In addition to the applications mentioned above, all of the format variations are useful in the support of non-standard bit-synchronous protocols.

## 18.6 HDLC

In addition to its support of SDLC communications, the 8044 also supports some of the capabilities of HDLC. The following remarks indicate the principal differences between SDLC and HDLC.

HDLC permits any number of bits in the information field, whereas SDLC requires a byte structure (multiple of 8 bits). The 8044 itself operates on byte boundaries, and thus it restricts fields to multiples of 8 bits.

HDLC provides functional extensions to SDLC: an unlimited address field is allowed, and extended frame number sequencing.

HDLC does not support operation in loop configurations.

## 18.7 SIU SPECIAL FUNCTION REGISTERS

The 8044 CPU communicates with and controls the SIU through hardware registers. These registers are accessed using direct addressing. The SIU special function registers (SIU SFRs) are of three types:

Control and Status Registers

Parameter Registers

ICE Support Registers

### 18.7.1 Control and Status Registers

There are three SIU Control and Status Registers:

Serial Mode Register (SMD)

Status/Command Register (STS)

### Send/Receive Count Register (NSNR)

The SMD, STS, and NSNR registers are all cleared by system reset. This assures that the SIU will power up in an idle state (neither receiving nor transmitting).

These registers and their bit assignments are described below (see also the More Details on Registers section).

### SMD: Serial Mode Register (byte-addressable)

|      |      |      |      |      |      |     |    |      |
|------|------|------|------|------|------|-----|----|------|
| Bit: | 7    | 6    | 5    | 4    | 3    | 2   | 1  | 0    |
|      | SCM2 | SCM1 | SCM0 | NRZI | LOOP | PFS | NB | NFCS |

The Serial Mode Register (Address C9H) selects the operational modes of the SIU. The 8044 CPU can both read and write SMD. The SIU can read SMD but cannot write to it. To prevent conflict between CPU and SIU access to SMD, the CPU should write SMD only when the Request To Send (RTS) and Receive Buffer Empty (RBE) bits (in the STS register) are both false (0). Normally, SMD is accessed only during initialization.

The individual bits of the Serial Mode Register are as follows:

| Bit # | Name | Description   |
|-------|------|---|
| SMD.0 | NFCS | No FCS field in the SDLC frame.   |
| SMD.1 | NB   | Non-Buffered mode. No control field in the SDLC frame.  |
| SMD.2 | PFS  | Pre-Frame Sync mode. In this mode, the 8044 transmits two bytes before the first flag of a frame, for DPLL synchronization. If NRZI is enabled, 00H is sent; otherwise, 55H is sent. In either case, 16 pre-frame transitions are guaranteed. |
| SMD.3 | LOOP | Loop configuration.   |
| SMD.4 | NRZI | NRZI coding option.   |
| SMD.5 | SCM0 | Select Clock Mode — Bit 0   |
| SMD.6 | SCM1 | Select Clock Mode — Bit 1   |
| SMD.7 | SCM2 | Select Clock Mode — Bit 2   |

The SCM bits decode as follows:

| SCM   | Clock Mode                   | Data Rate (Bits/sec)* |
|-------|------------------------------|-----------------------|
| 2 1 0 | Externally clocked           | 0-2.4M**              |
| 0 0 0 | Undefined                    |                       |
| 0 0 1 | Undefined                    |                       |
| 0 1 0 | Self clocked, timer overflow | 244-62.5K             |
| 0 1 1 | Undefined                    |                       |
| 1 0 0 | Self clocked, external 16x   | 0-375K                |

| SCM |   | Clock Mode | Data Rate<br>(Bits/sec)*     |
|-----|---|------------|------------------------------|
| 2   | 1 | 0          |                              |
| 1   | 0 | 1          | Self clocked, external 32K   |
| 1   | 1 | 0          | Self clocked, internal fixed |
| 1   | 1 | 1          | Self clocked, internal fixed |

\*Based on a 12 Mhz crystal frequency

\*\*0-1M bps in loop configuration

#### STS: Status/Command Register (bit-addressable)

|      |     |     |     |    |     |     |    |     |
|------|-----|-----|-----|----|-----|-----|----|-----|
| Bit: | 7   | 6   | 5   | 4  | 3   | 2   | 1  | 0   |
|      | TBF | RBE | RTS | SI | BOV | OPB | AM | RBP |

The Status/Command Register (Address C8H) provides operational control of the SIU by the 8044 CPU, and enables the SIU to post status information for the CPU's access. The SIU can read STS, and can alter certain bits, as indicated below. The CPU can both read and write STS asynchronously. However, 2-cycle instructions that access STS during both cycles ('JBC /B, REL' and 'MOV /B,C.') should not be used, since the SIU may write to STS between the two CPU accesses.

The individual bits of the Status/Command Register are as follows:

| Bit # | Name | Description   |
|-------|------|---|
| STS.0 | RBP  | Receive Buffer Protect. Inhibits writing of data into the receive buffer. In AUTO mode, RBP forces an RNR response instead of an RR.  |
| STS.1 | AM   | AUTO Mode/Addressed Mode. Selects AUTO mode where AUTO mode is allowed. If NB is true, (=1), the AM bit selects the addressed mode. AM may be cleared by the SIU.                                     |
| STS.2 | OPB  | Optional Poll Bit. Determines whether the SIU will generate an AUTO response to an optional poll (UP with P=0). OPB may be set or cleared by the SIU.   |
| STS.3 | BOV  | Receive Buffer Overrun. BOV may be set or cleared by the SIU.   |
| STS.4 | SI   | SIU Interrupt. This is one of the five interrupt sources to the CPU. The vector location = 23H. SI may be set by the SIU. It should be cleared by the CPU before returning from an interrupt routine. |
| STS.5 | RTS  | Request To Send. Indicates that the 8044 is ready to transmit or is trans-  |

mitting. RTS may be read or written by the CPU. RTS may be read by the SIU, and in AUTO mode may be written by the SIU.

STS.6 RBE Receive Buffer Empty. RBE can be thought of as Receive Enable. RBE is set to one by the CPU when it is ready to receive a frame, or has just read the buffer, and to zero by the SIU when a frame has been received.

STS.7 TBF Transmit Buffer Full. Written by the CPU to indicate that it has filled the transmit buffer. TBF may be cleared by the SIU.

#### NSNR: Send/Receive Count Register (bit-addressable)

|      |     |     |     |     |     |     |     |     |
|------|-----|-----|-----|-----|-----|-----|-----|-----|
| Bit: | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |
|      | NS2 | NS1 | NS0 | SES | NR2 | NR1 | NR0 | SER |

The Send/Receive Count Register (Address D8H) contains the transmit and receive sequence numbers, plus tally error indications. The SIU can both read and write NSNR. The 8044 CPU can both read and write NSNR asynchronously. However, 2-cycle instructions that access NSNR during both cycles ('JBC /B, REL', and 'MOV /B,C.') should not be used, since the SIU may write to NSNR between the two 8044 CPU accesses.

The individual bits of the Send/Receive Count Register are as follows:

| Bit #  | Name | Description  |
|--------|------|--|
| NSNR.0 | SER  | Receive Sequence Error: NS (P) ≠ NR (S)                      |
| NSNR.1 | NR0  | Receive Sequence Counter—Bit 0                               |
| NSNR.2 | NR1  | Receive Sequence Counter—Bit 1                               |
| NSNR.3 | NR2  | Receive Sequence Counter—Bit 2                               |
| NSNR.4 | SES  | Send Sequence Error: NR (P) ≠ NS (S) and NR (P) ≠ NS (S) + 1 |
| NSNR.5 | NS0  | Send Sequence Counter — Bit 0                                |
| NSNR.6 | NS1  | Send Sequence Counter — Bit 1                                |
| NSNR.7 | NS2  | Send Sequence Counter — Bit 2                                |

#### 18.7.2 Parameter Registers

There are eight parameter registers that are used in connection with SIU operation. All eight registers may be read or written by the 8044 CPU. RFL and RCB are normally loaded by the SIU.

The eight parameter registers are as follows:

#### STAD: Station Address Register (byte-addressable)

The Station Address register (Address CEH) contains the station address. To prevent access conflict, the CPU should access STAD only when the SIU is idle (RTS=0 and RBE=0). Normally, STAD is accessed only during initialization.

#### TBS: Transmit Buffer Start Address Register (byte-addressable)

The Transmit Buffer Start address register (Address DCH) points to the location in on-chip RAM for the beginning of the I-field of the frame to be transmitted. The CPU should access TBS only when the SIU is not transmitting a frame (when TBF=0).

#### TBL: Transmit Buffer Length Register (byte-addressable)

The Transmit Buffer Length register (Address DBH) contains the length (in bytes) of the I-field to be transmitted. A blank I-field (TBL=0) is valid. The CPU should access TBL only when the SIU is not transmitting a frame (when TBF=0).

NOTE: The transmit and receive buffers are not allowed to "wrap around" in the on-chip RAM. A "buffer end" is automatically generated if address 191 (BFH) is reached.

#### TCB: Transmit Control Byte Register (byte-addressable)

The Transmit Control Byte register (Address DAH) contains the byte which is to be placed in the control field of the transmitted frame, during NON-AUTO mode transmission. The CPU should access TCB only when the SIU is not transmitting a frame (when TBF=0). The N<sub>S</sub> and N<sub>R</sub> counters are not used in the NON-AUTO mode.

#### RBS: Receive Buffer Start Address Register (byte-addressable)

The Receive Buffer Start address register (Address CCH) points to the location in on-chip RAM where the beginning of the I-field of the frame being received is to be stored. The CPU should write RBS only when the SIU is not receiving a frame (when RBE=0).

#### RBL: Receive Buffer Length Register (byte-addressable)

The Receive Buffer Length register (Address CBH) contains the length (in bytes) of the area in on-chip

RAM allocated for the received I-field. RBL=0 is valid. The CPU should write RBL only when RBE=0.

#### RFL: Receive Field Length Register (byte-addressable)

The Received Field Length register (Address CDH) contains the length (in bytes) of the received I-field that has just been loaded into on-chip RAM. RFL is loaded by the SIU. RFL=0 is valid. RFL should be accessed by the CPU only when RBE=0.

#### RCB: Receive Control Byte Register (byte-addressable)

The Received Control Byte register (Address CAH) contains the control field of the frame that has just been received. RCB is loaded by the SIU. The CPU can only read RCB, and should only access RCB when RBE=0.

### 18.73 ICE Support Registers

The 8044 In-Circuit Emulator (ICE-44) allows the user to exercise the 8044 application system and monitor the execution of instructions in real time.

The emulator operates with Intel's Intellec® development system. The development system interfaces with the user's 8044 system through an in-cable buffer box. The cable terminates in a 8044 pin-compatible plug, which fits into the 8044 socket in the user's system. With the emulator plug in place, the user can exercise his system in real time while collecting up to 255 instruction cycles of real-time data. In addition, he can single-step the program.

Static RAM is available (in the in-cable buffer box) to emulate the 8044 internal and external program memory and external data memory. The designer can display and alter the contents of the replacement memory in the buffer box, the internal data memory, and the internal 8044 registers, including the SFRs.

Among the SIU SFRs are the following registers that support the operation of the ICE:

#### DMA CNT: DMA Count Register (byte-addressable)

The DMA Count register (Address CFH) indicates the number of bytes remaining in the information block that is currently being used.

#### FIFO: Three-Byte (byte-addressable)

The Three-Byte FIFO (Address DDH, DEH, and DFH) is used between the eight-bit shift register and the information buffer when an information block is received.



### SIUST: SIU State Counter (byte-addressable)

The SIU State Counter (Address D9H) reflects the state of the internal logic which is under SIU control. Therefore, care must be taken not to write into this register.

The SIUST register can serve as a helpful aid to determine which field of a receive frame that the SIU expects next. The table below will help in debugging 8044 reception problems.

#### SIUST

##### VALUE FUNCTION

01H Waiting for opening flag.

08H Waiting for address field.

10H Waiting for control field.

18H Waiting for first byte of I field. This state is only entered if a FCS is expected. It pushes the received byte onto the top of the FIFO.

20H Waiting for second byte of I field. This state always follows state 18H.

28H Waiting for I field byte. This state can be en-

tered from state 20H or from states 01H, 08H, or 10H depending upon the SIU's mode configuration. (Each time a byte is received, it is pushed onto the top of the FIFO and the byte at the bottom is put into memory. For no FCS formatted frames, the FIFO is collapsed into a single register).

30H Waiting for the closing flag after having overflowed the receive buffer. Note that even if the receive frame overflows the assigned receive buffer length, the FCS is still checked.

Examples of SIUST status sequences for different frame formats are shown below. Note that status changes after acceptance of the received field byte.

### 18.8 OPERATION

The SIU is initialized by a reset signal (on pin 9), followed by write operations to the SIU SFRs. Once initialized, the SIU can function in AUTO mode or NON-AUTO mode. Details are given below.

Table 18-1. SIUST Status Sequences

Example 1:

Frame Format

SIUST Value

| (Idle) | F  | A  | C  | I  |    |    | FCS | F  |
|--------|----|----|----|----|----|----|-----|----|
| 01     | 01 | 08 | 10 | 18 | 20 | 28 | 28  | 01 |

Example 2:

Frame Format

SIUST Value

| (Idle) | F  | A  | I  |    | FCS | F  |
|--------|----|----|----|----|-----|----|
| 01     | 01 | 08 | 18 | 20 | 28  | 01 |

Example 3:

Frame Format

SIUST Value

| (Idle) | F  |    | I  |    | FCS | F  |
|--------|----|----|----|----|-----|----|
| 01     | 01 | 18 | 20 | 28 | 28  | 01 |

Example 4:

Frame Format

SIUST Value

| (Idle) | F  | A  | I  | F  |
|--------|----|----|----|----|
| 01     | 01 | 08 | 28 | 01 |

Example 5:

Frame Format

SIUST Value

| (Idle) | F  | I  | F  |
|--------|----|----|----|
| 01     | 01 | 28 | 01 |

Example 6:

Frame Format

SIUST Value

| (Idle) | F  | I  | I OVERFLOW | FCS | F  |
|--------|----|----|------------|-----|----|
| 01     | 01 | 18 | 20         | 30  | 01 |

#### Frame Option

| NFCS | NB | AM |
|------|----|----|
| 0    | 0  | 1  |

|   |   |   |
|---|---|---|
| 0 | 1 | 1 |
|---|---|---|

|   |   |   |
|---|---|---|
| 0 | 1 | 0 |
|---|---|---|

|   |   |   |
|---|---|---|
| 1 | 1 | 1 |
|---|---|---|

|   |   |   |
|---|---|---|
| 1 | 1 | 0 |
|---|---|---|

|   |   |   |
|---|---|---|
| 0 | 1 | 0 |
|---|---|---|

### 18.8.1 Initialization

Figure 18-6 is the SIU. Registers SMD, STS, and NSNR are cleared by reset. This puts the 8044 into an idle state—neither receiving nor transmitting. The following registers must be initialized before the 8044 leaves the idle state:

**STAD**—to establish the 8044's SDLC station address.

**SMD**—to configure the 8044 for the proper operating mode.

**RBS, RBL**—to define the area in RAM allocated for the Receive Buffer.

**TBS, TBL**—to define the area in RAM allocated for the Transmit Buffer.

Once these registers have been initialized, the user may write to the STS register to enable the SIU to leave the idle state, and to begin transmits and/or receives.

Setting RBE to 1 enables the SIU for receive. When RBE = 1, the SIU monitors the received data stream for a flag pattern. When a flag pattern is found, the SIU enters Receive mode and receives the frame.

Setting RTS to 1 enables the SIU for transmit. When RTS = 1, the SIU monitors the received data stream for a GA pattern (loop configuration) or waits for a CTS

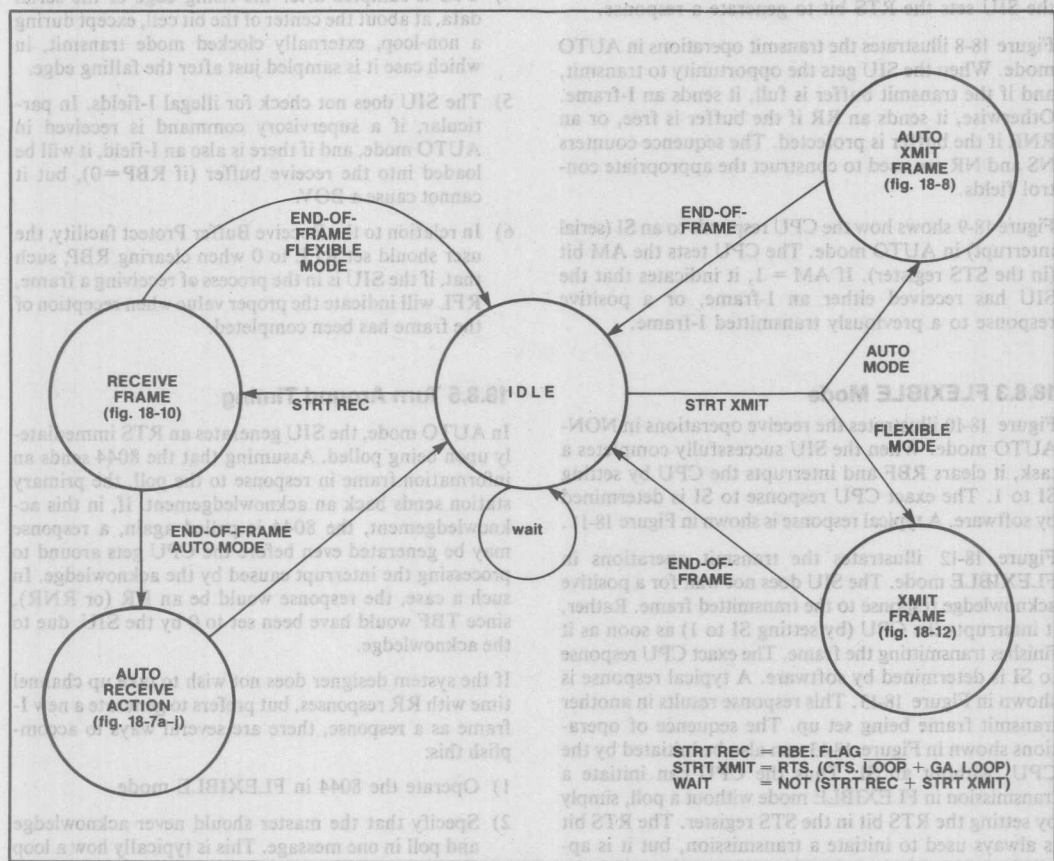


Figure 18-6. SIU State Diagram

(non-loop configuration). When the GA or CTS arrives, the SIU enters Transmit mode and transmits a frame.

In AUTO mode, the SIU sets RTS to enable automatic transmissions of appropriate responses.

### 18.8.2 AUTO Mode

Figure 18-7 illustrates the receive operations in AUTO mode. The overall operation is shown in Figure 18-7a. Particular cases are illustrated in Figures 18-7b through 18-7j. If any Unnumbered Command other than UP is received, the AM bit is cleared and the SIU responds as if in the FLEXIBLE mode, by interrupting the CPU for supervision. This will also happen if a BOV or SES condition occurs. If the received frame contains a poll, the SIU sets the RTS bit to generate a response.

Figure 18-8 illustrates the transmit operations in AUTO mode. When the SIU gets the opportunity to transmit, and if the transmit buffer is full, it sends an I-frame. Otherwise, it sends an RR if the buffer is free, or an RNR if the buffer is protected. The sequence counters NS and NR are used to construct the appropriate control fields.

Figure 18-9 shows how the CPU responds to an SI (serial interrupt) in AUTO mode. The CPU tests the AM bit (in the STS register). If AM = 1, it indicates that the SIU has received either an I-frame, or a positive response to a previously transmitted I-frame.

### 18.8.3 FLEXIBLE Mode

Figure 18-10 illustrates the receive operations in NON-AUTO mode. When the SIU successfully completes a task, it clears RBF and interrupts the CPU by setting SI to 1. The exact CPU response to SI is determined by software. A typical response is shown in Figure 18-11.

Figure 18-12 illustrates the transmit operations in FLEXIBLE mode. The SIU does not wait for a positive acknowledge response to the transmitted frame. Rather, it interrupts the CPU (by setting SI to 1) as soon as it finishes transmitting the frame. The exact CPU response to SI is determined by software. A typical response is shown in Figure 18-13. This response results in another transmit frame being set up. The sequence of operations shown in Figure 18-13 can also be initiated by the CPU, without an SI. Thus the CPU can initiate a transmission in FLEXIBLE mode without a poll, simply by setting the RTS bit in the STS register. The RTS bit is always used to initiate a transmission, but it is applied to the RTS pin only when a non-loop configuration is used.

### 18.8.4 8044 Data Link Particulars

The following facts should be noted:

- 1) In a non-loop configuration, one or two bits are transmitted before the opening flag. This is necessary for NRZI synchronization.
- 2) In a non-loop configuration, one to eight extra dribble bits are transmitted after the closing flag. These bits are a zero followed by ones.
- 3) In a loop configuration, when a GA is received and the 8044 begins transmitting, the sequence is 0111111010111110... (FLAG, 1, FLAG, ADDRESS, etc.). The first flag is created from the GA. The second flag begins the message.
- 4) CTS is sampled after the rising edge of the serial data, at about the center of the bit cell, except during a non-loop, externally clocked mode transmit, in which case it is sampled just after the falling edge.
- 5) The SIU does not check for illegal I-fields. In particular, if a supervisory command is received in AUTO mode, and if there is also an I-field, it will be loaded into the receive buffer (if RBP=0), but it cannot cause a BOV.
- 6) In relation to the Receive Buffer Protect facility, the user should set RFL to 0 when clearing RBP, such that, if the SIU is in the process of receiving a frame, RFL will indicate the proper value when reception of the frame has been completed.

### 18.8.5 Turn Around Timing

In AUTO mode, the SIU generates an RTS immediately upon being polled. Assuming that the 8044 sends an information frame in response to the poll, the primary station sends back an acknowledgement. If, in this acknowledgement, the 8044 is polled again, a response may be generated even before the CPU gets around to processing the interrupt caused by the acknowledge. In such a case, the response would be an RR (or RNR), since TBF would have been set to 0 by the SIU, due to the acknowledge.

If the system designer does not wish to take up channel time with RR responses, but prefers to generate a new I-frame as a response, there are several ways to accomplish this:

- 1) Operate the 8044 in FLEXIBLE mode.
- 2) Specify that the master should never acknowledge and poll in one message. This is typically how a loop system operates, with the poll operation confined to the UP command. This leaves plenty of time for the

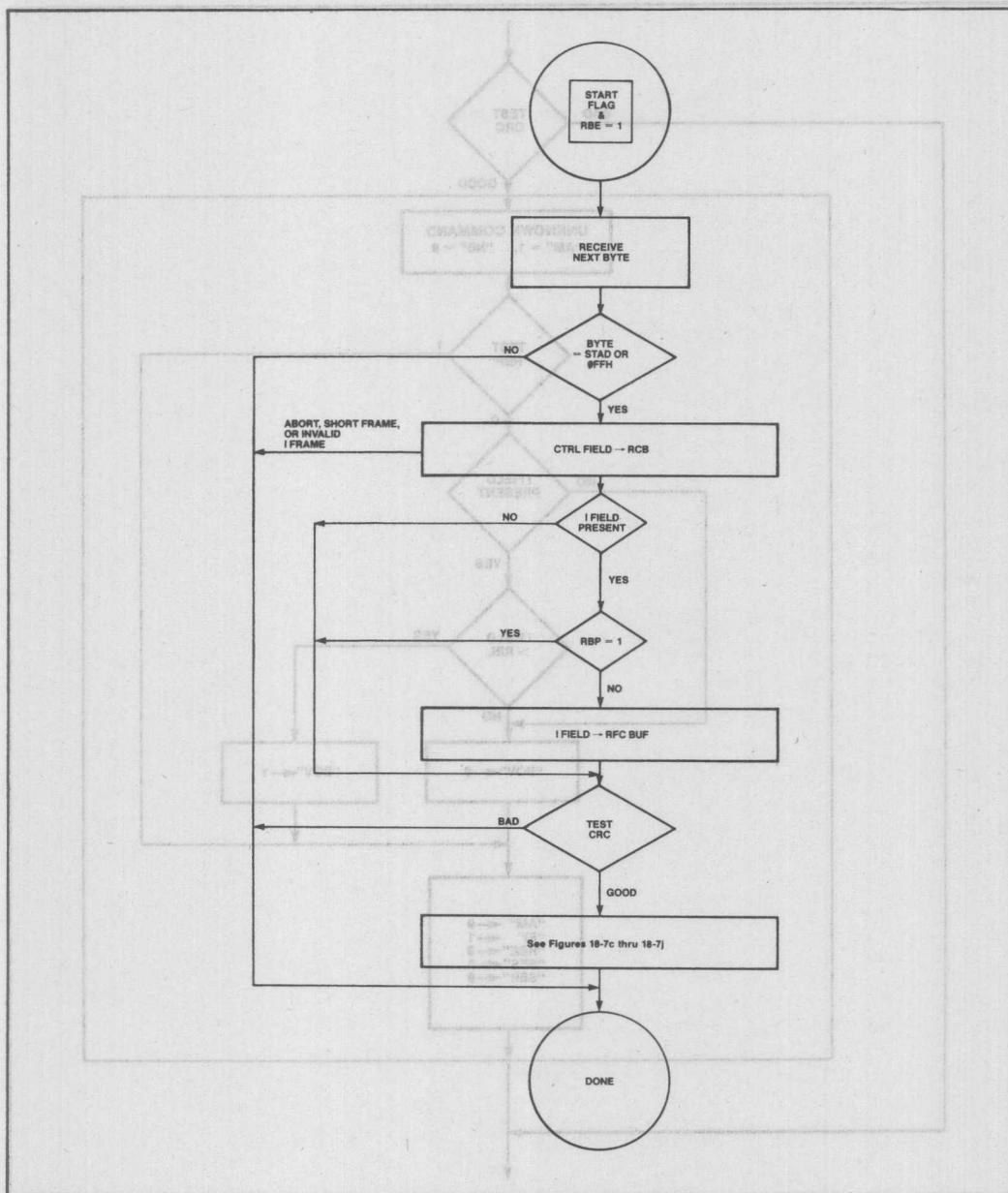


Figure 18-7a. SIU AUTO Mode Receive Flowchart—General



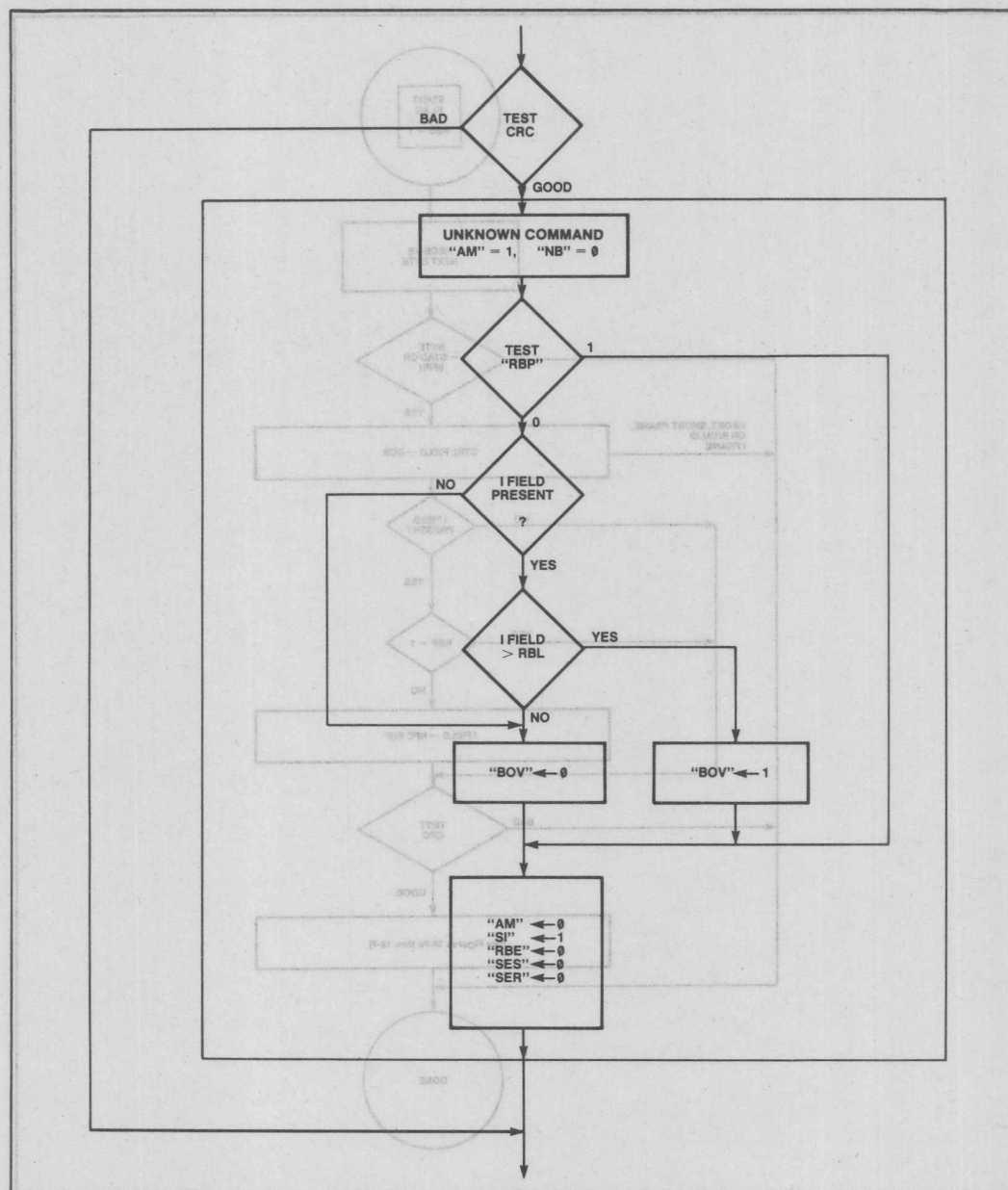


Figure 18-7b. SIU AUTO Mode Receive Flowchart—Unknown Command

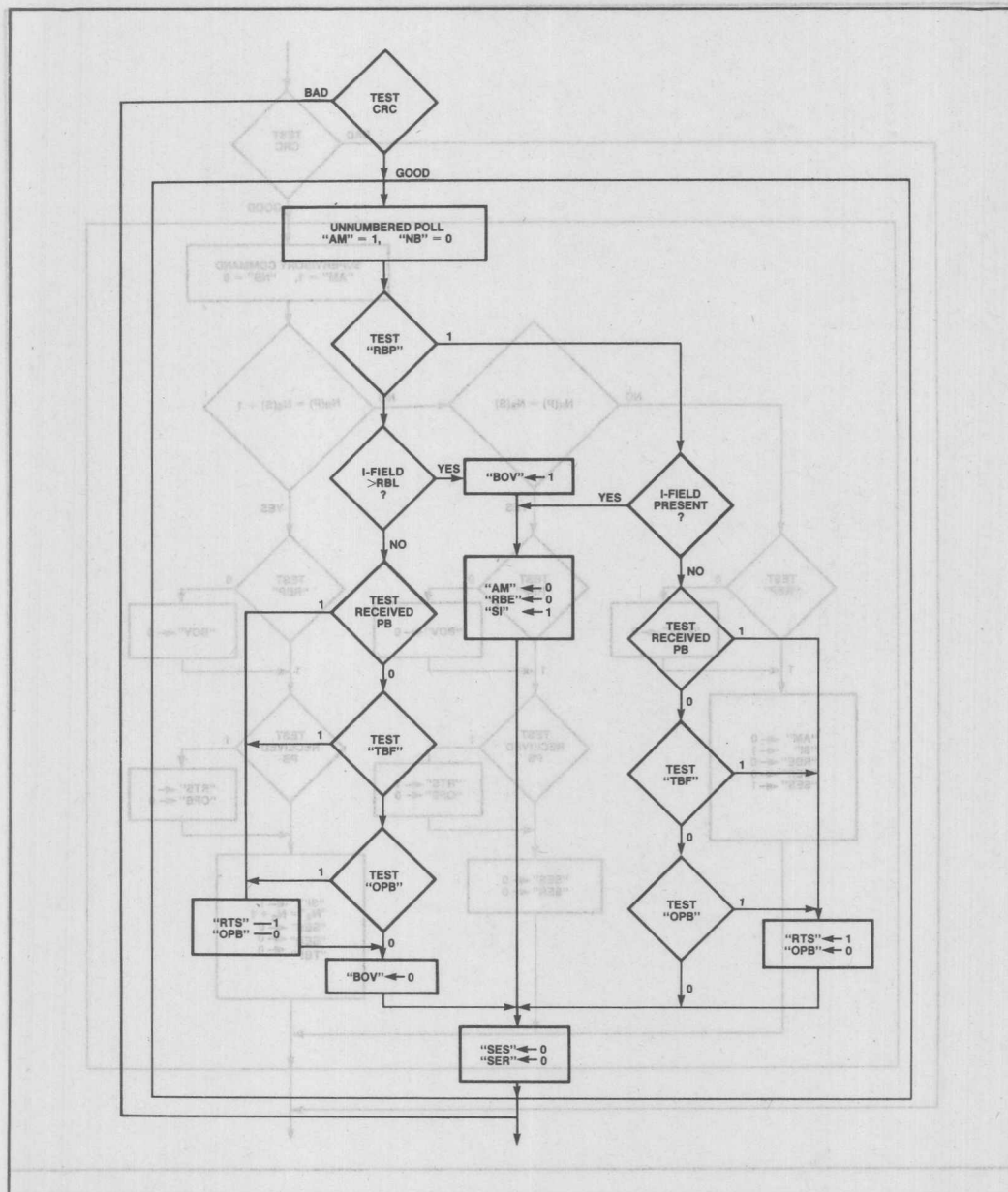


Figure 18-7d. SIU AUTO Mode Receive Flowchart—Supervisory Command

Figure 18-7c. SIU AUTO Mode Receive Flowchart—Unnumbered Poll

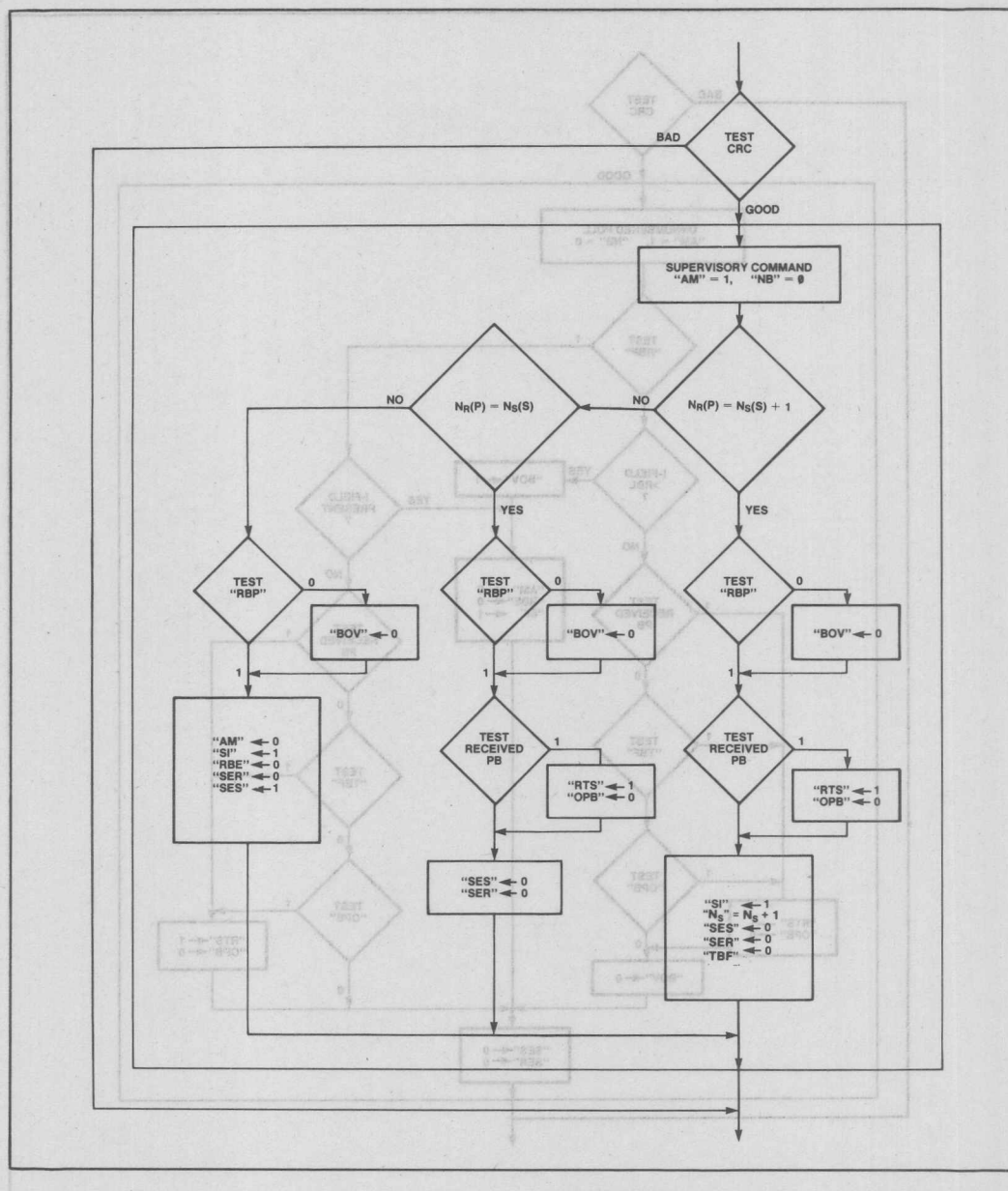


Figure 18-7d. SIU AUTO Mode Receive Flowchart—Supervisory Command

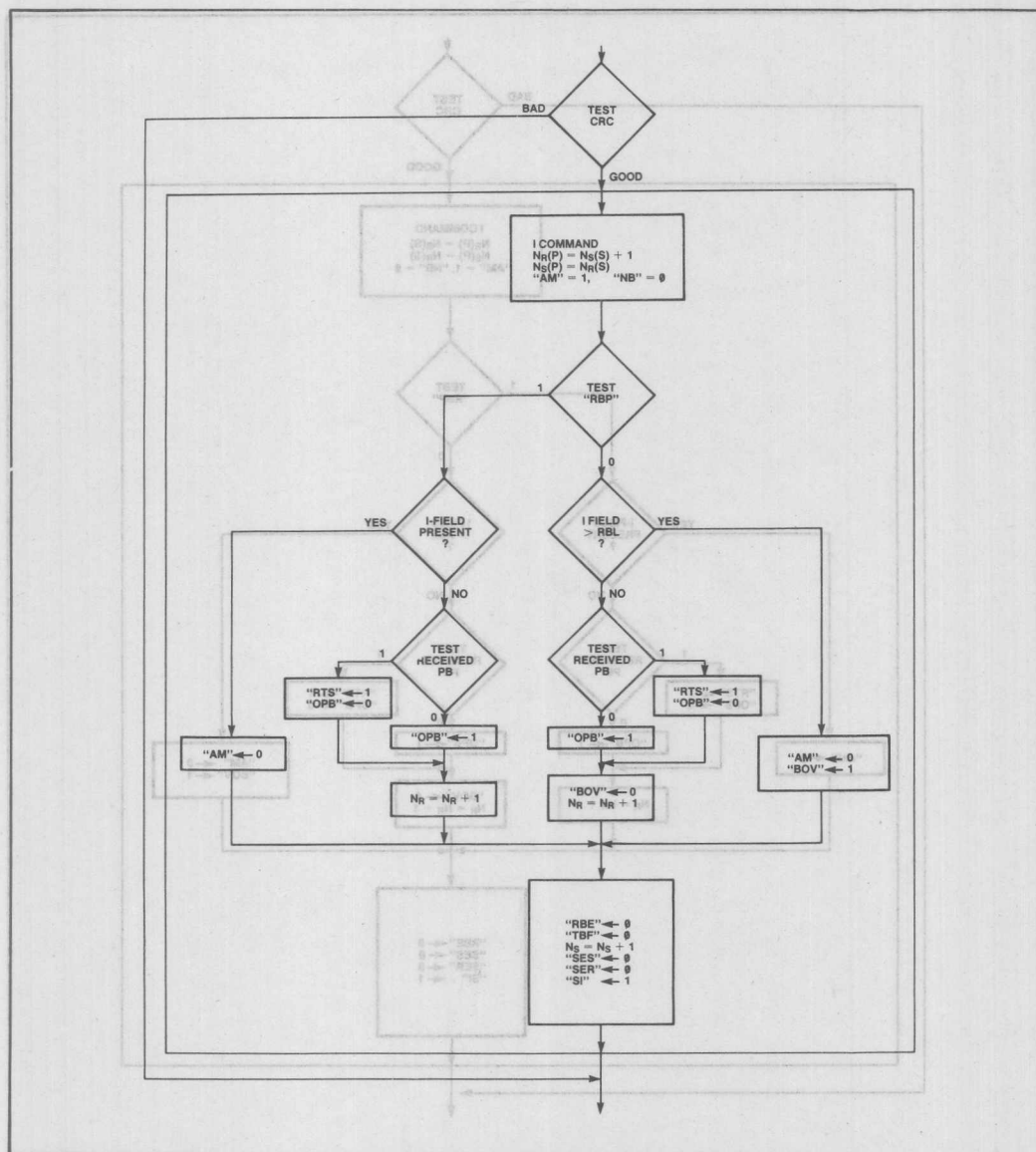


Figure 18-7e. SIU AUTO Mode Receive Flowchart—I Command: Prior Transmitted I-Field Confirmed, Current Received I-Field in Sequence



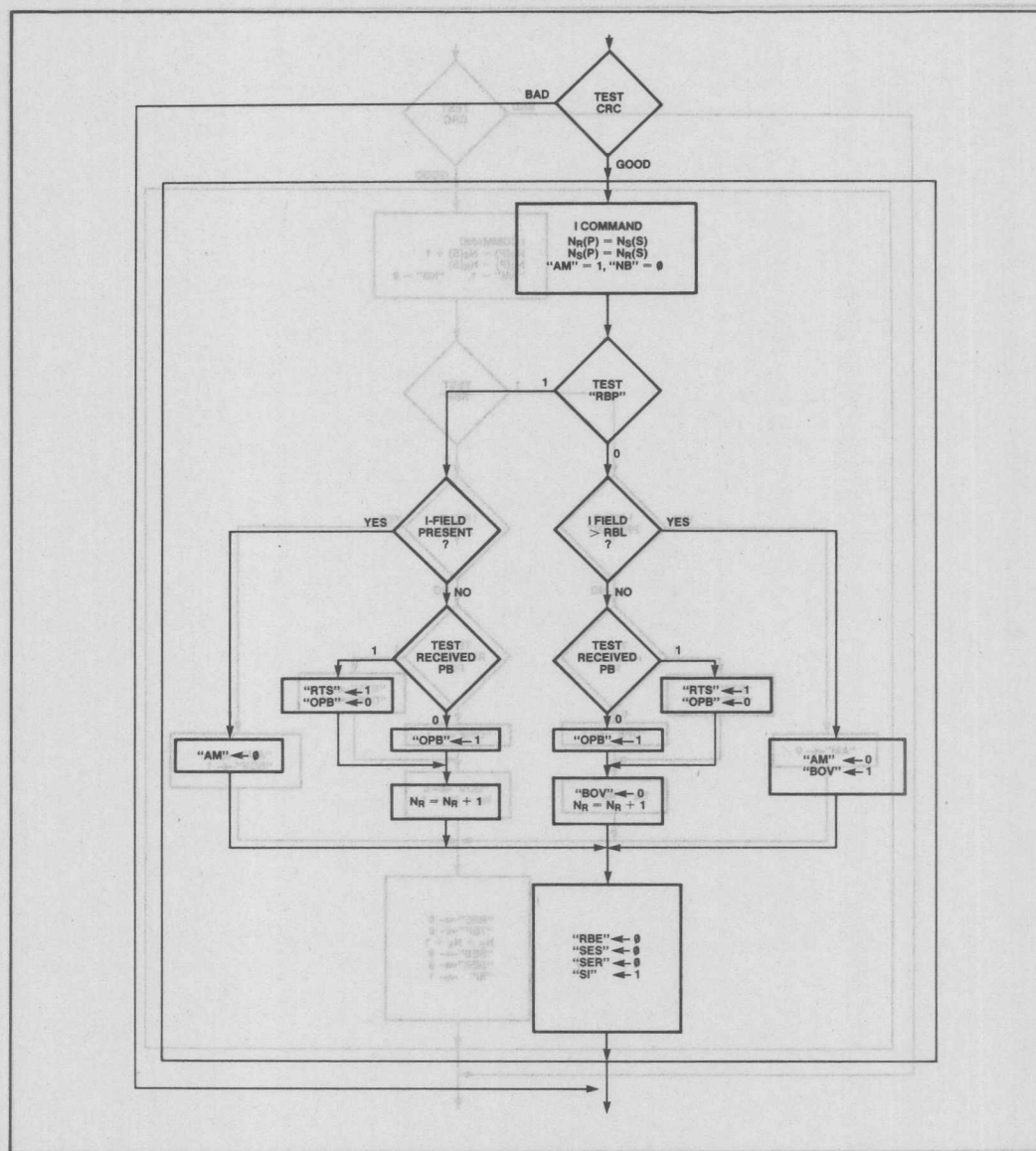


Figure 18-71. SIU AUTO Mode Receive Flowchart—I Command: Prior Transmitted I-Field Not Confirmed, Current Received I-Field in Sequence

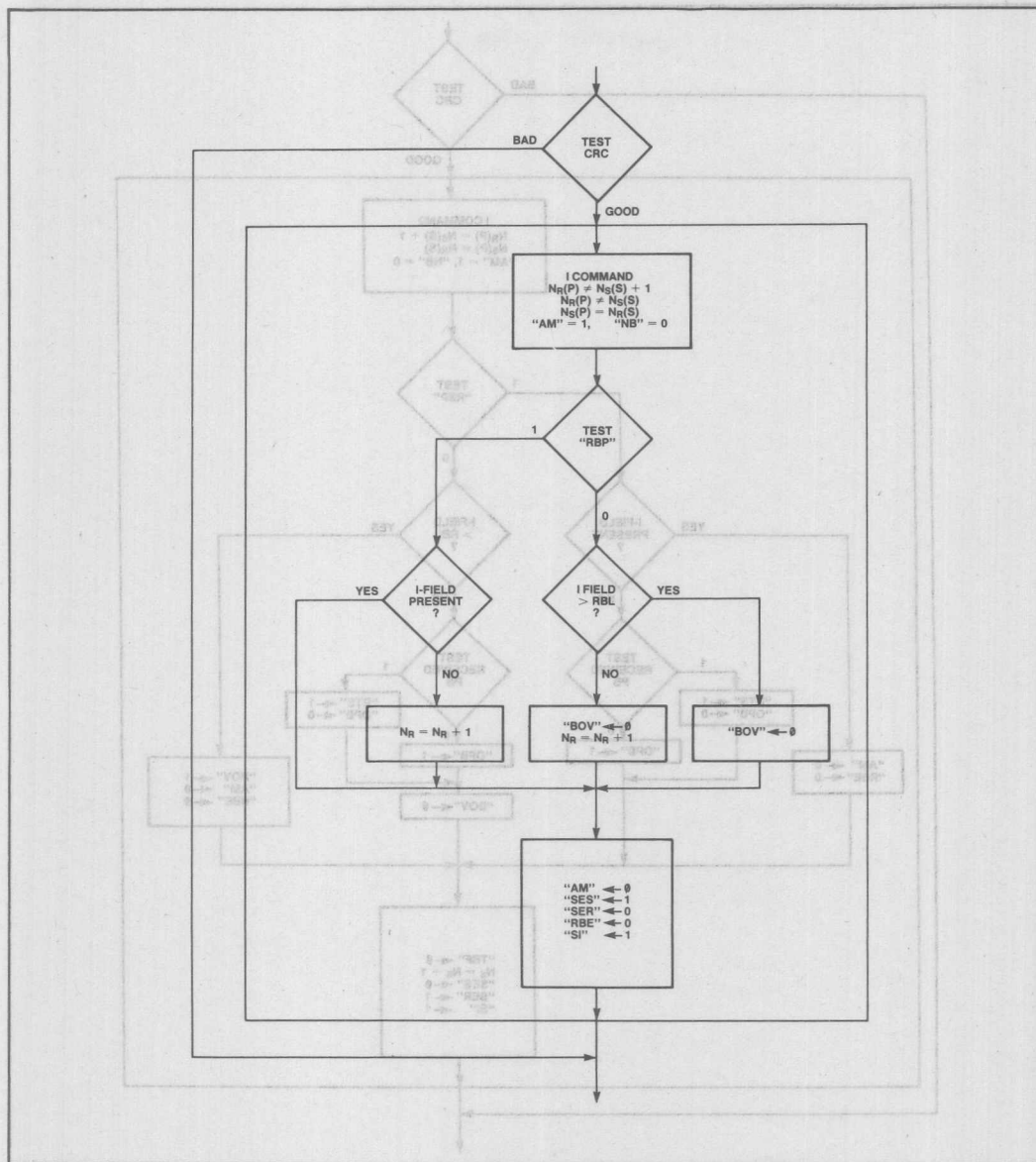


Figure 18-7g. SIU AUTO Mode Receive Flowchart—I Command: Sequence Error Send, Current Received I-Field in Sequence

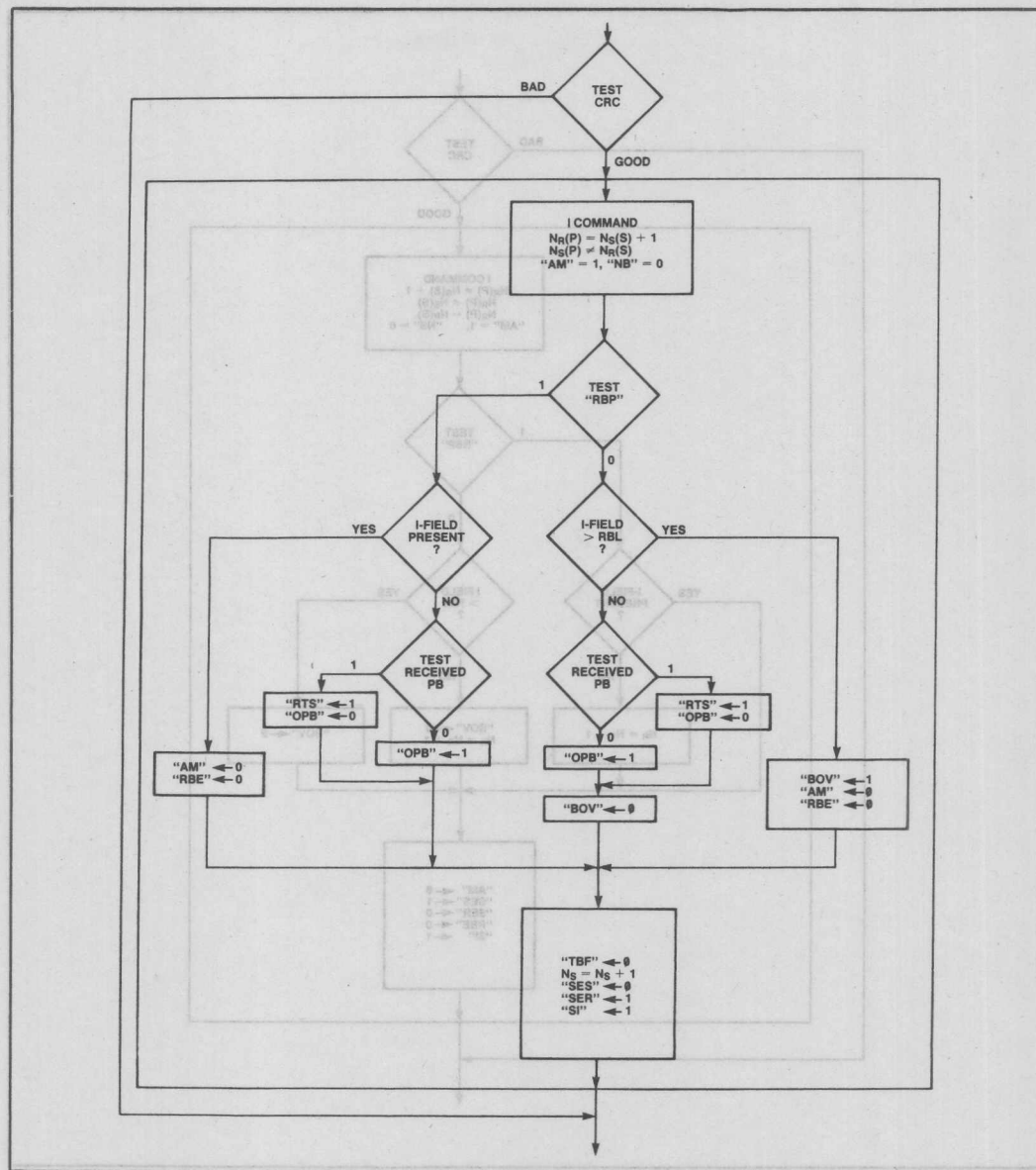


Figure 18-7h. SIU AUTO Mode Receive Flowchart—I Command: Prior Transmitted I-Field Confirmed Sequence Error Receive

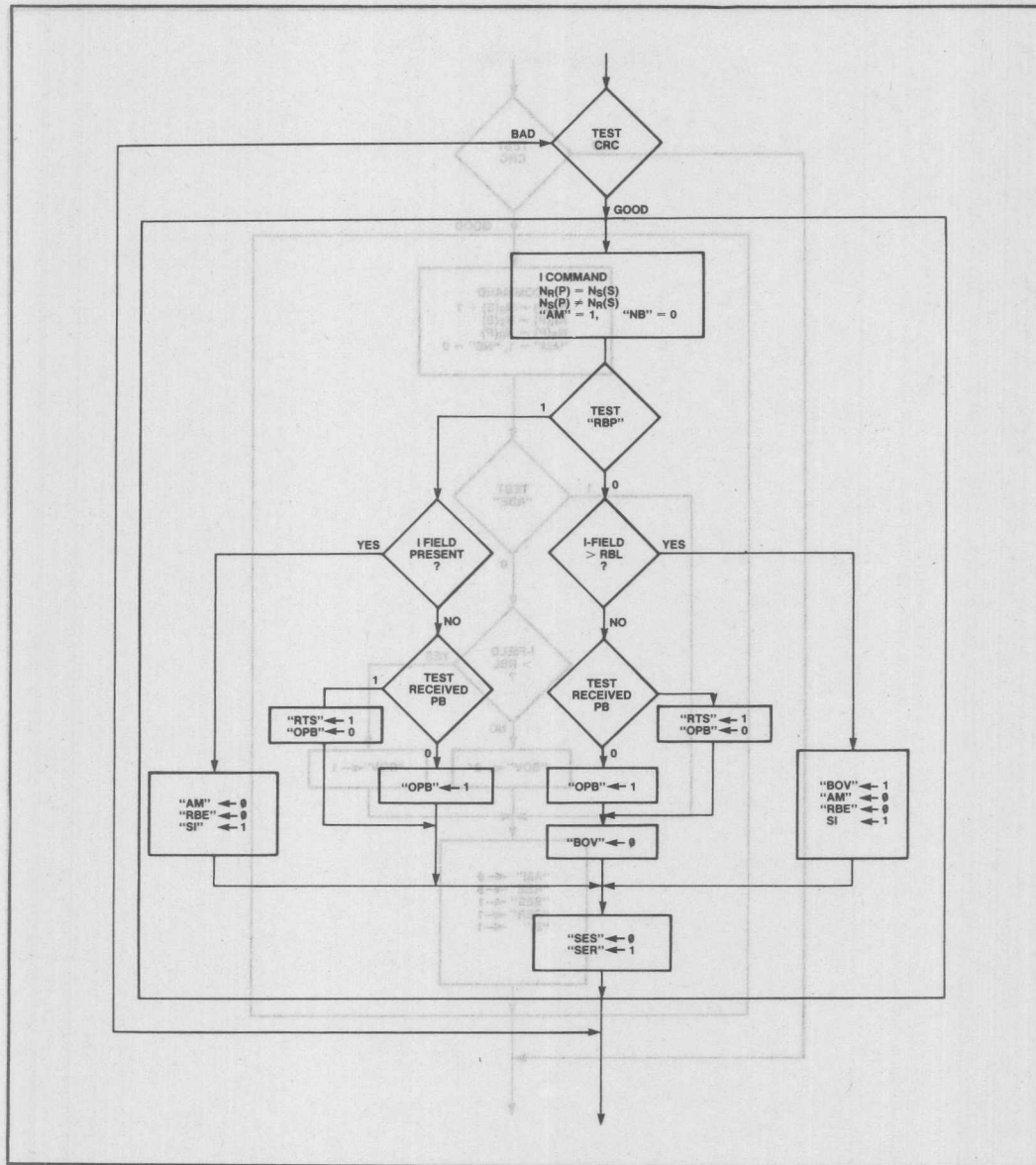


Figure 18-71. SIU AUTO Mode Receive Flowchart—I Command: Prior Transmitted I-Field Not Confirmed, Sequence Error Receive



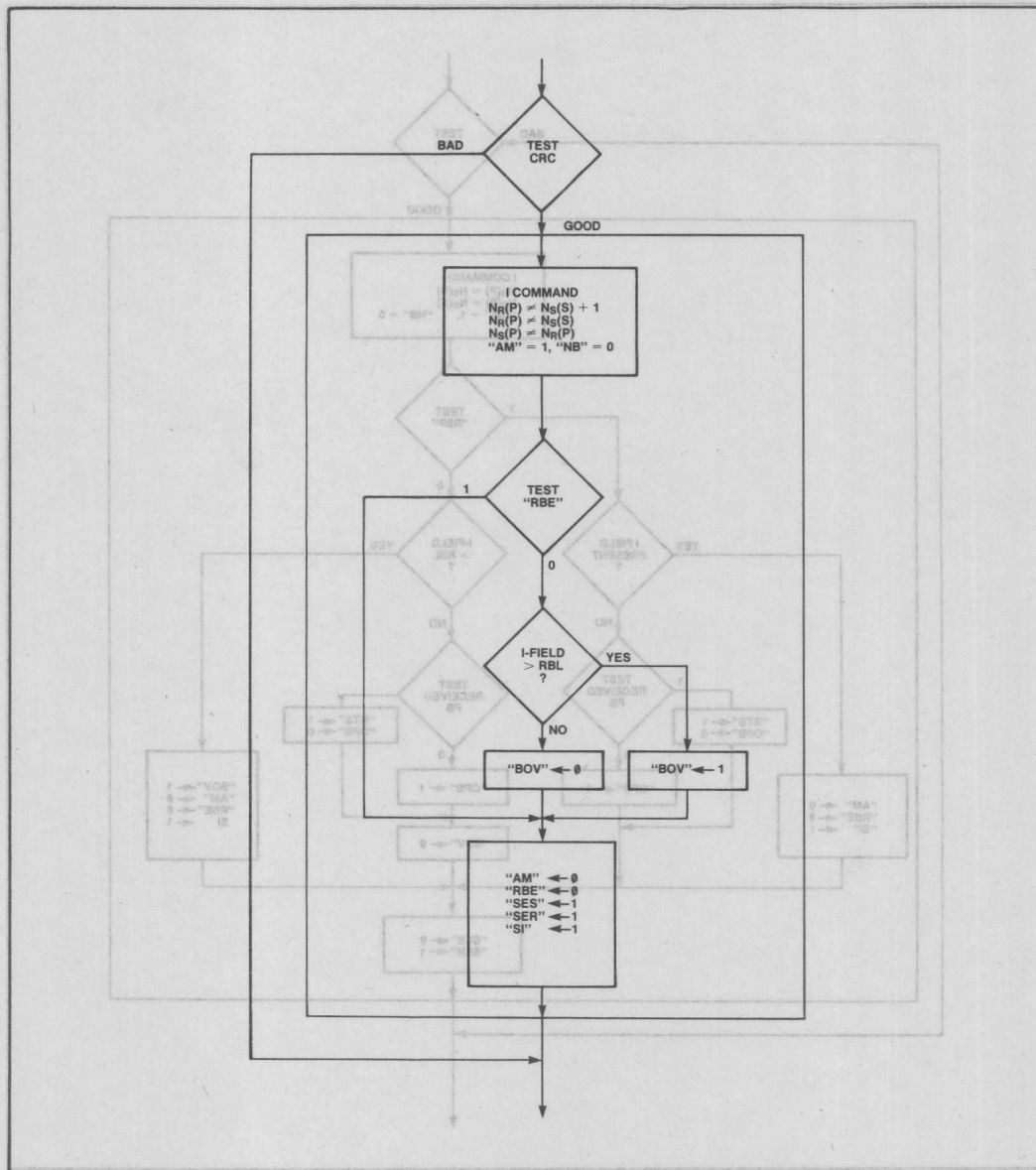


Figure 18-7j. SIU SUTO Mode Receive Flowchart—I Command: Sequence Error Send and Sequence Error Receive

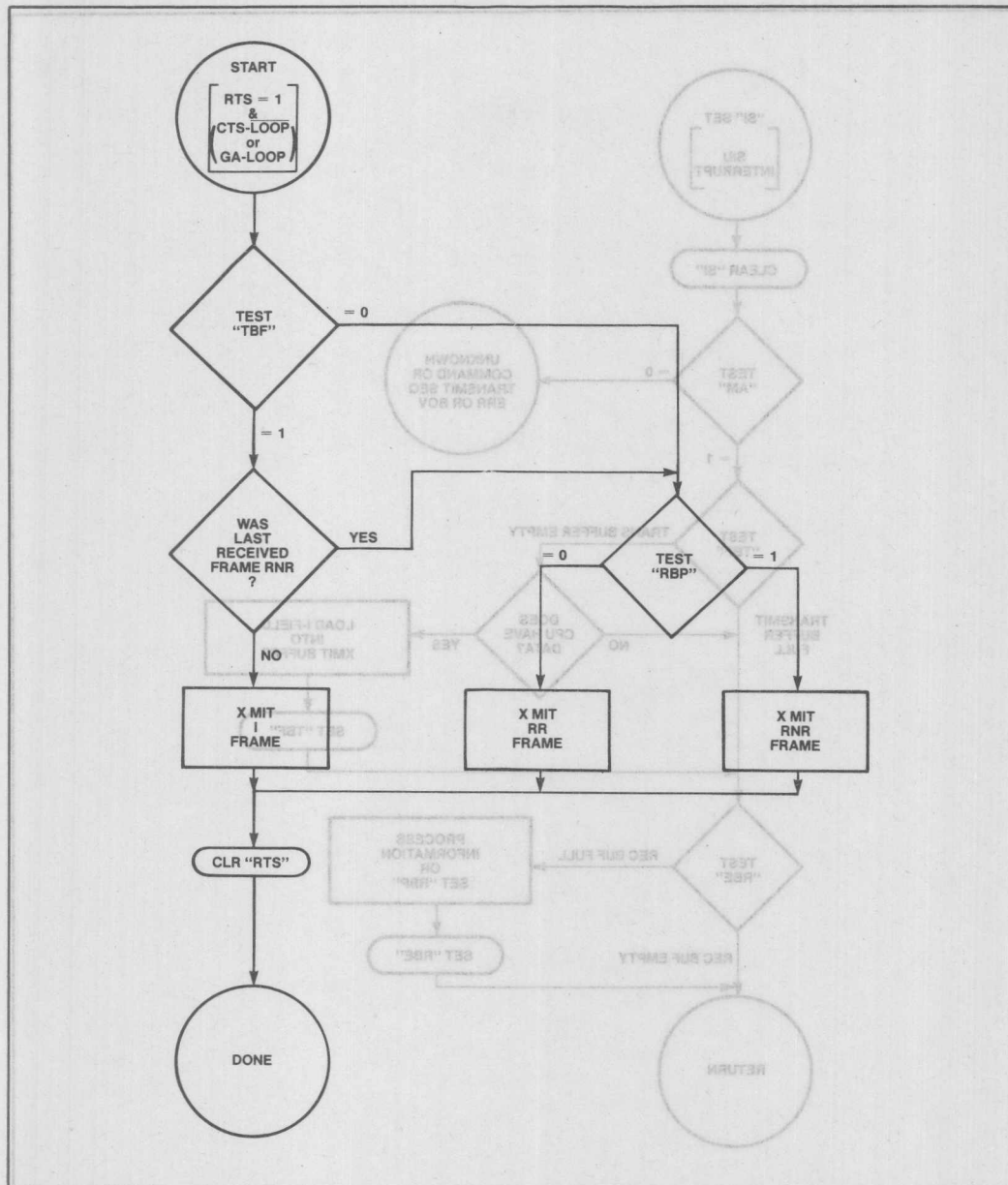


Figure 18-8. SIU AUTO Mode Transmit Flowchart

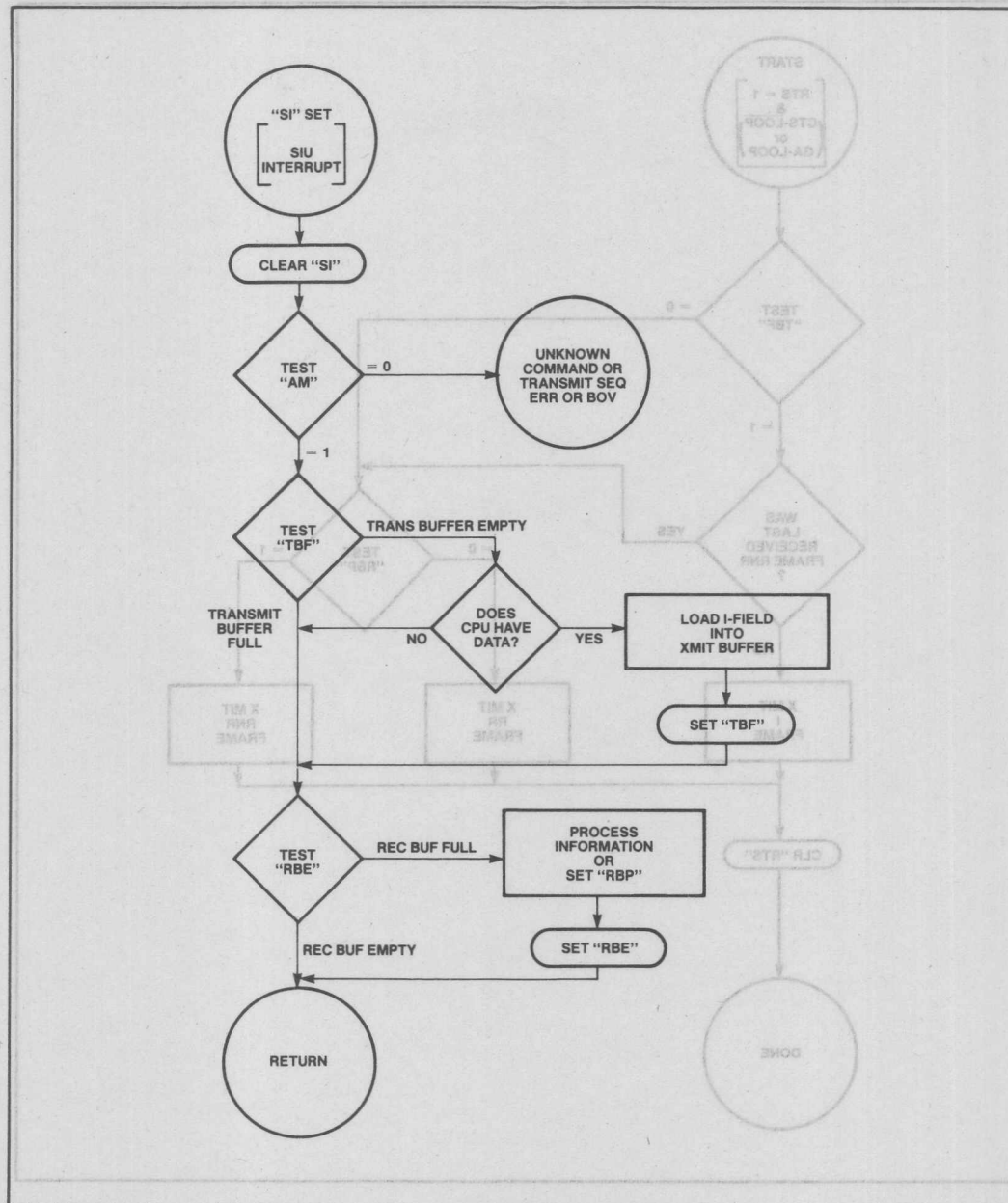


Figure 18-9. AUTO Mode Response to "SI"

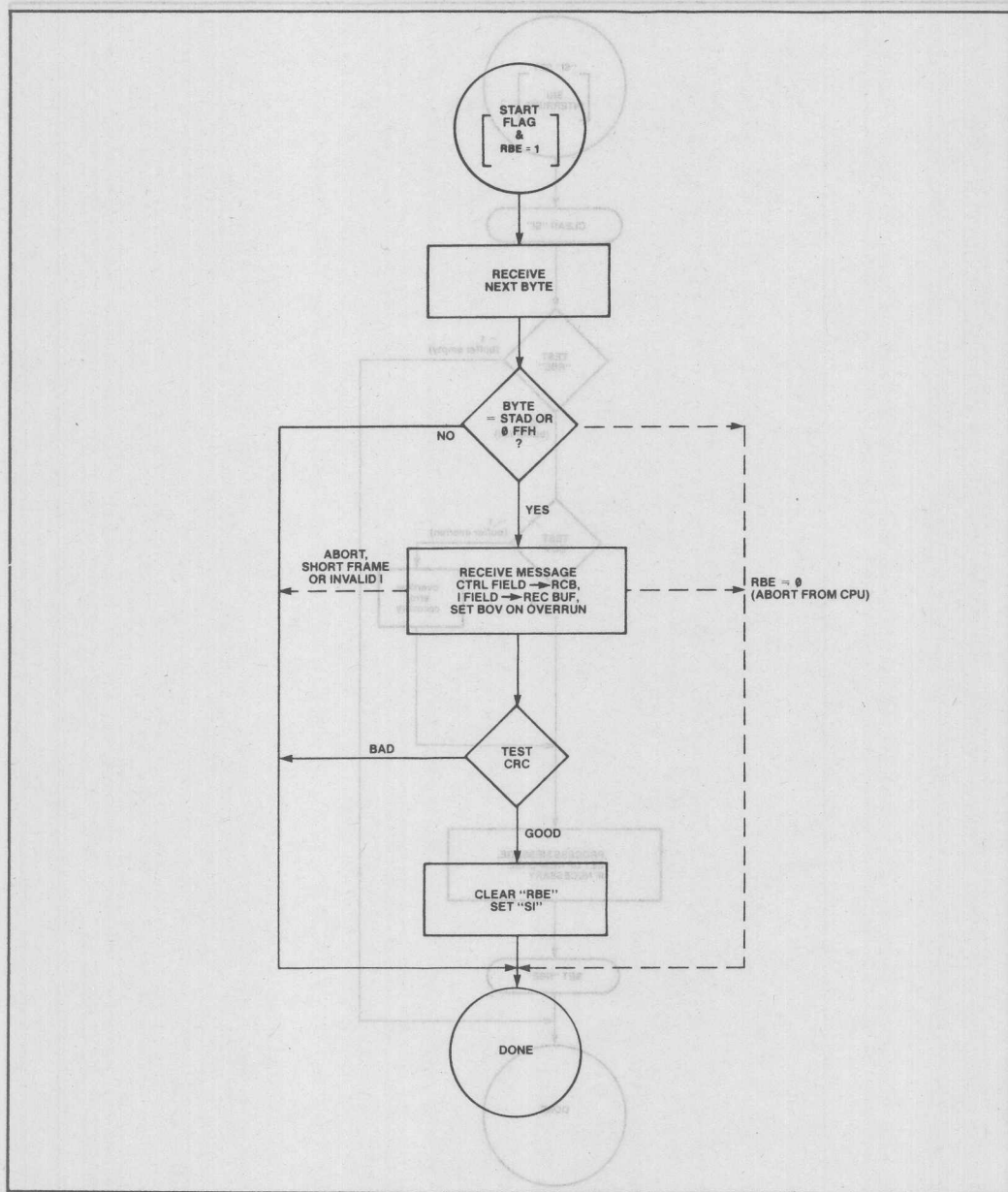


Figure 18-10. SIU FLEXIBLE Mode Receive Flowchart



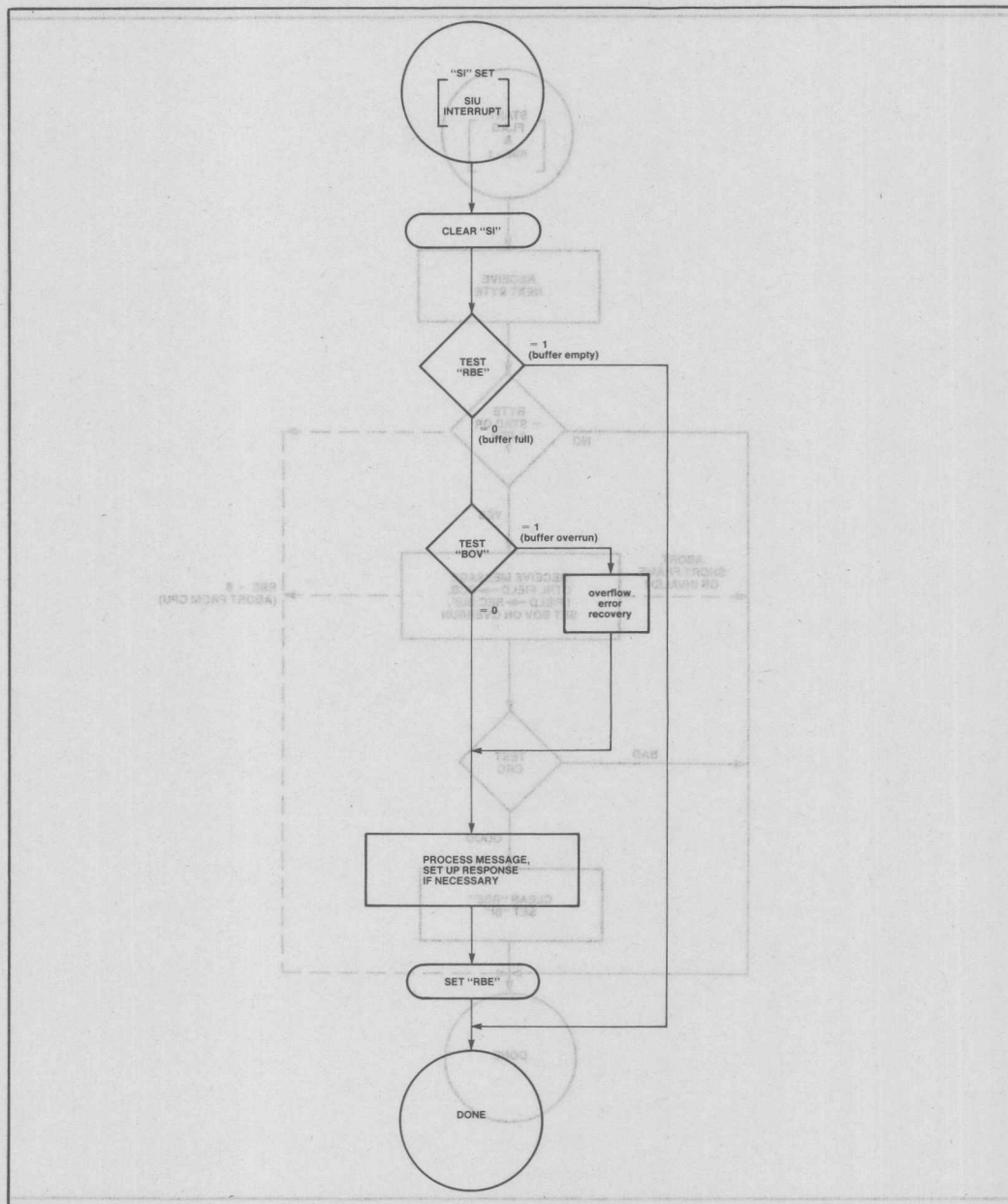


Figure 18-11. FLEXIBLE Mode Response to Receive "SI"

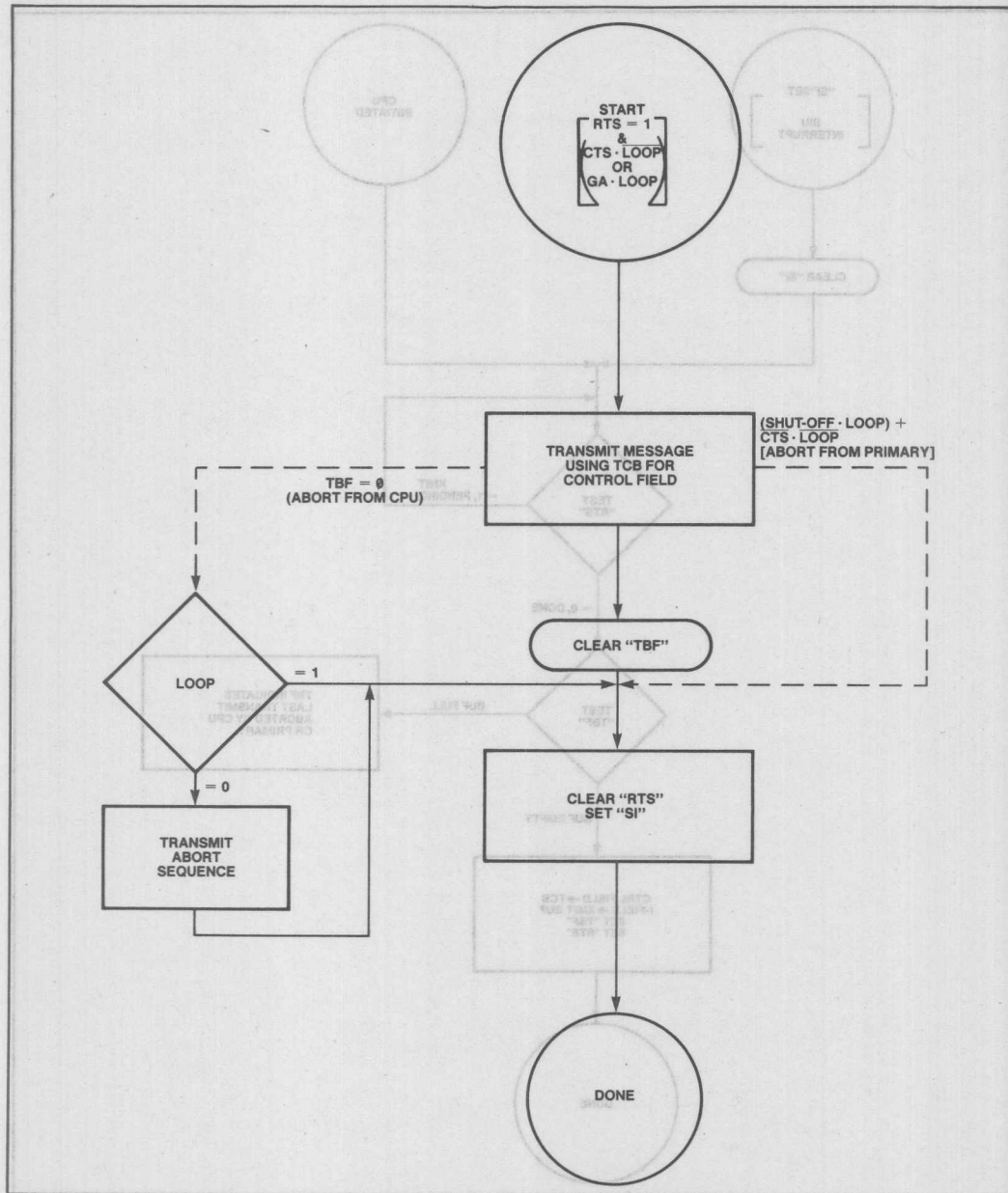


Figure 18-12. SIU FLEXIBLE Mode Transmit Flowchart

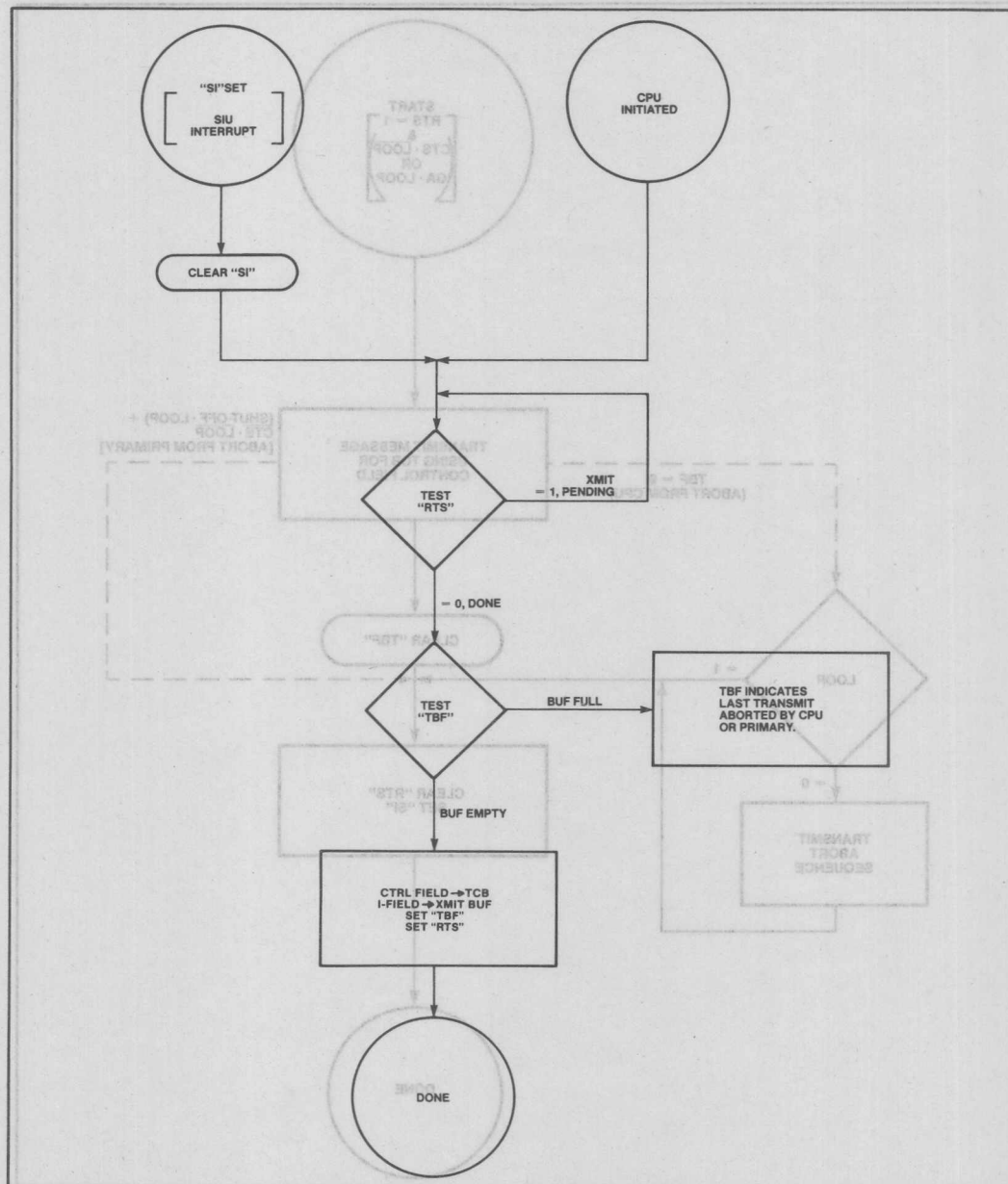


Figure 18-13. FLEXIBLE Mode Response to Transmit "SI"

8044 to get its transmit buffer loaded with new information after an acknowledgment.

- 3) The 8044 CPU can clear RTS. This will prevent a response from being sent, or abort it if it is already in progress. A system using external RTS/CTS handshaking could use a one-shot to delay RTS or CTS, thereby giving the CPU more time to disable the response.

## 18.9 MORE DETAILS ON SIU HARDWARE

The SIU divides functionally into two sections—a bit processor (BIP) and a byte processor (BYP)—sharing some common timing and control logic. As shown in Figure 18-14, the BIP operates between the serial port pins and the SIU bus, and performs all functions necessary to transmit/receive a byte of data to/from the serial data stream. These operations include shifting, NRZI encoding/decoding, zero insertion/deletion, and FCS generation/checking. The BYP manipulates bytes of data to perform message formatting, and other transmitting and receiving functions. It operates between the SIU bus (SIB) and the 8044's internal bus (IB). The interface between the SIU and the CPU involves an interrupt and some locations in on-chip RAM space which are managed by the BYP.

The maximum possible data rate for the serial port is limited to 1/2 the internal clock rate. This limit is imposed by both the maximum rate of DMA to the on-chip RAM, and by the requirements of synchronizing to an external clock. The internal clock rate for an 8044 running on a 12 MHz crystal is 6 MHz. Thus the maximum 8044 serial data rate is 3 MHz. This data rate drops down to 2.4 MHz when time is allowed for external clock synchronization.

### 18.9.1 The Bit Processor

In the asynchronous (self clocked) modes the clock is extracted from the data stream using the on-chip digital phase-locked-loop (DPLL). The DPLL requires a clock input at 16 times the data rate. This  $16 \times$  clock may originate from SCLK, Timer 1 Overflow, or PH2 (one half the oscillator frequency). The extra divide-by-two described above allows these sources to be treated alternatively as  $32 \times$  clocks.

The DPLL is a free-running four-bit counter running off the  $16 \times$  clock. When a transition is detected in the receive data stream, a count is dropped (by suppressing the carry-in) if the current count value is greater than 8. A count is added (by injecting a carry into the second stage rather than the first) if the count is less than 8. No

adjustment is made if the transition occurs at the count of 8. In this manner the counter locks in on the point at which transitions in the data stream occur at the count of 8, and a clock pulse is generated when the count overflows to 0.

In order to perform NRZI decoding, the NRZI decoder compares each bit of input data to the previous bit. There are no clock delays in going through the NRZI decoder.

The zero insert/delete circuitry (ZID) performs zero insertion/deletion, and also detects flags, GA's (Go-Ahead's), and aborts (same as GA's) in the data stream. The pattern 1111110 is detected as an early GA, so that the GA may be turned into a flag for loop mode transmission.

The shut-off detector monitors the receive data stream for a sequence of eight zeros, which is a shut-off command for loop mode transmissions. The shut-off detector is a three-bit counter which is cleared whenever a one is found in the receive data stream. Note that the ZID logic could not be used for this purpose, because the receive data must be monitored even when the ZID is being used for transmission.

As an example of the operation of the bit processor, the following sequence occurs in relation to the receive data:

- 1) RXD is sampled by SCLK, and then synchronized to the internal processor clock (IPC).
- 2) If the NRZI mode is selected, the incoming data is NRZI decoded.
- 3) When receiving other than the flag pattern, the ZID deletes the '0' after 5 consecutive '1's (during transmission this zero is inserted). The ZID locates the byte boundary for the rest of the circuitry. The ZID deletes the '0's by preventing the SR (shift register) from receiving a clocking pulse.
- 4) The FCS (which is a function of the data between the flags—not including the flags) is initialized and started at the detection of the byte boundary at the end of the opening flag. The FCS is computed each bit boundary until the closing flag is detected. Note that the received FCS has gone through the ZID during transmission.

### 18.9.2 The Byte Processor

Figure 18-15 is a block diagram of the byte processor (BYP). The BYP contains the registers and controllers necessary to perform the data manipulations associated with SDLC communications. The BYP registers may be read or written by the CPU over the 8044's internal bus



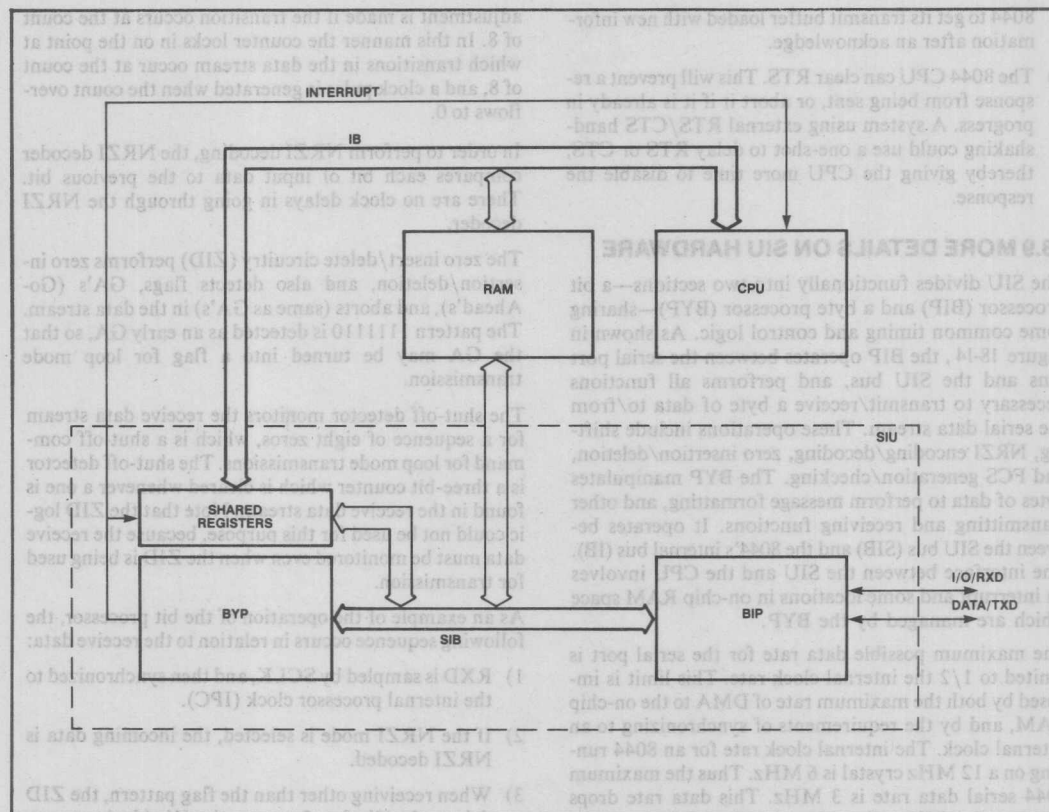


Figure 18-14. The Bit and Byte Processors

(IB), using standard 8044 hardware register operations. The 8044 register select PLA controls these operations. Three of the BYP registers connect to the IB through the IBS, a sub-bus which also connects to the CPU interrupt control registers.

Simultaneous access of a register by both the IB and the SIB is prevented by timing. In particular, RAM access is restricted to alternate internal processor cycles for the CPU and the SIU, in such a way that collisions do not occur.

As an example of the operation of the byte processor, the following sequence occurs in relation to the receive data:

- 1) Assuming that there is an address field in the frame, the BYP takes the station address from the register file into temporary storage. After the opening flag,

the next field (the address field) is compared to the station address in the temporary storage. If a match occurs, the operation continues.

- 2) Assuming that there is a control field in the frame, the BYP takes the next byte and loads it into the RCB register. The RCB register has the logic to update the NSNR register (increment receive count, set SES and SER flags, etc.).
- 3) Assuming that there is an information field, the next byte is dumped into RAM at the RBS location. The DMA CNT (RBL at the opening flag) is loaded from the DMA CNT register into the RB register and decremented. The RFL is then loaded into the RB register, incremented, and stored back into the register file.

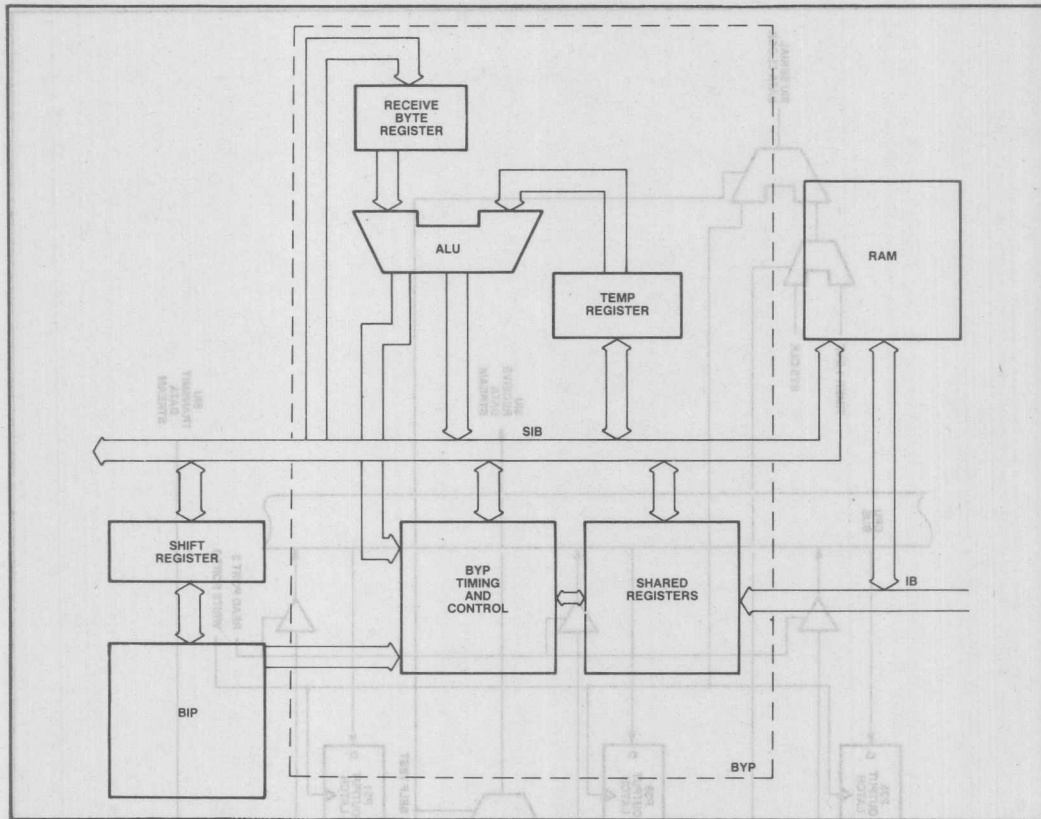


Figure 18-15. The Byte Processor

- 4) This process continues until the DMA CNT reaches zero, or until a closing flag is received. Upon either event, the BYP updates the status, and, if the CRC is good, the NSNR register.

### 18.10 DIAGNOSTICS

An SIU test mode has been provided, so that the on-chip CPU can perform limited diagnostics on the SIU. The test mode utilizes the output latches for P3.0 and P3.1 (pins 10 and 11). These port 3 pins are not useful as output ports, since the pins are taken up by the serial port functions. Figure 18-16 shows the signal routing associated with the SIU test mode.

Writing a 0 to P3.1 enables the serial test mode (P3.1 is set to 1 by reset). In test mode the P3.0 bit is mapped

into the received data stream, and the 'write port 3' control signal is mapped into the SCLK path in place of T1. Thus, in test mode, the CPU can send a serial data stream to the SIU by writing to P3.0. The transmit data stream can be monitored by reading P3.1. Each successive bit is transmitted from the SIU by writing to any bit in Port 3, which generates SCLK.

In test mode, the P3.0 and P3.1 pins are placed in a high voltage, high impedance state. When the CPU reads P3.0 and P3.1 the logic level applied to the pin will be returned. In the test mode, when the CPU reads P3.1, the transmit data value will be returned, not the voltage on the pin. The transmit data remains constant for a bit time. Writing to P3.0 will result in the signal being outputted for a short period of time. However, since the signal is not latched, P3.0 will quickly return to a high voltage, high impedance state.

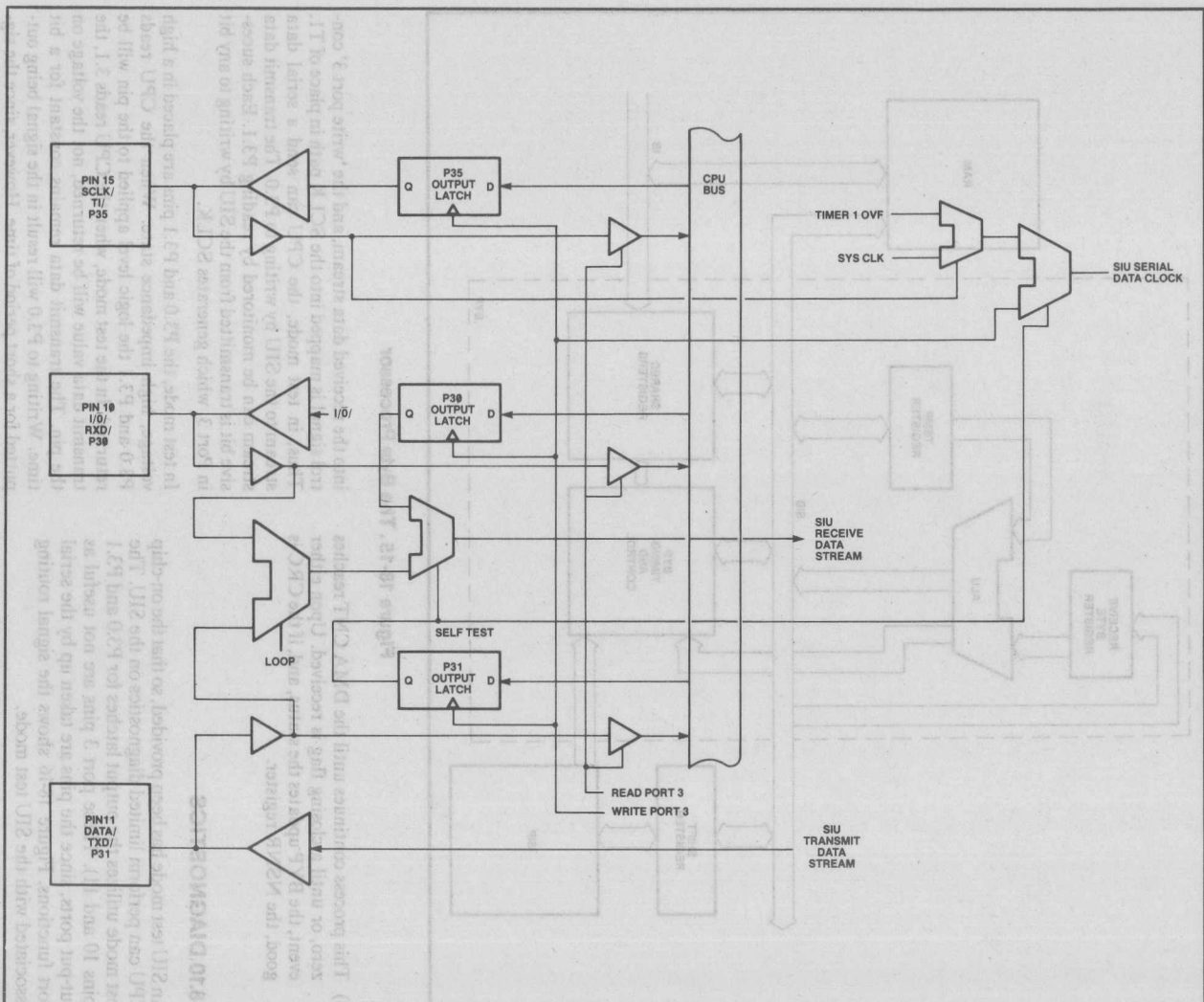


Figure 18-16. SIU Test Mode

The serial test mode is disabled by writing a 1 to P3.1. Care must be taken that a 0 is never written to P3.1 in the course of normal operation, since this causes the test mode to be entered.

Figure 18-17 is an example of a simple program segment that can be imbedded into the user's diagnostic program. That example shows how to put the 8044 into "Loop-back mode" to test the basic transmitting and receiving functions of the SIU.

Loop-back mode is functionally equivalent to a hardware connection between pins 10 and 11 on the 8044.

In this example, the 8044 CPU plays the role of the primary station. The SIU is in the AUTO mode. The CPU sends the SIU a supervisory frame with the poll bit set and an RNR command. The SIU responds with a supervisory frame with the poll bit set and an RR command.

The operation proceeds as follows:

Interrupts are disabled, and the self test mode is enabled by writing a zero to P3.1. This establishes P3.0 as the data path from the CPU to the SIU. CTS (clear-to-send) is enabled by writing a zero to P1.7. The station address is initialized by writing 08AH into the STAD (station address register).

The SIU is configured for receive operation in the clocked mode and in AUTO mode. The CPU then transmits a supervisory frame. This frame consists of an

opening flag, followed by the station address, a control field indicating that this is a supervisory frame with an RNR command, and then a closing flag.

Each byte of the frame is transmitted by writing that byte into the A register and then calling the subroutine XMIT8. Two additional SCLKs are generated to guarantee that the last bits in the frame have been clocked into the SIU. Finally the CPU reads the status register (STS). If the operation has proceeded correctly, the status will be 072H. If it is not, the program jumps to the ERROR loop and terminates.

The SIU generates an SI (SIU interrupt) to indicate that it has received a frame. The CPU clears this interrupt, and then begins to monitor the data stream that is being generated by the SIU in response to what it has received. As each bit arrives (via P3.1), it is moved into the accumulator, and the CPU compares the byte in the accumulator with 07EH, which is the opening flag. When a match occurs, the CPU identifies this as byte boundary, and thereafter processes the information byte-by-byte.

The CPU calls the RCV8 subroutine to get each byte into the accumulator. The CPU performs compare operations on (successively) the station address, the control field (which contains the RR response), and the closing flag. If any of these do not compare, the program jumps to the ERROR loop. If no error is found, the program jumps to the DONE loop.

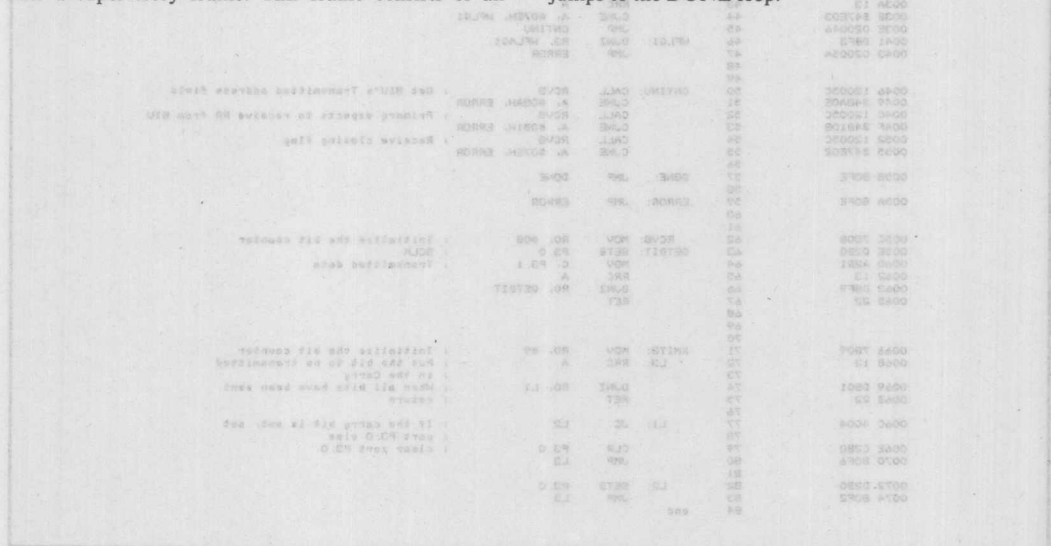


Figure 18-17. Loop-Back Mode Software



# MCS-51 MACRO ASSEMBLER DATA

1818-11 MCS-51 MACRO ASSEMBLER V2.0  
OBJECT MODULE PLACED IN: F1:DATA.OBJ  
ASSEMBLER INVOKED BY: asm51: f1: data.man device (44)

| LOC  | OBJ    | LINE | SOURCE  |
|------|--------|------|---|
|      |        | 1    |   |
|      |        | 2    |   |
| 0000 | 75C800 | 3    | INIT: MOV STS.#00H  |
| 0003 | C2B1   | 4    | CLR P3.1 ; Enable self test mode                                      |
| 0005 | C297   | 5    | CLR P1.7 ; Enable CTS   |
| 0007 | 75CEBA | 6    | MOV STAD.#8AH ; Initialize address                                    |
|      |        | 7    |   |
|      |        | 8    | ; CONFIGURE RECEIVE OPERATION   |
|      |        | 9    |   |
| 000A | 75DB6A | 10   | MOV NSNR.#6AH ; NS(S)=3, SES=0, NR(S)=5, SER=0                        |
| 000D | 75C901 | 11   | MOV SMD.#01H ; NFCS=1   |
| 0010 | 75CB22 | 12   | MOV STS.#0C2H ; TBF=1, RBE=1, AM=1                                    |
|      |        | 13   |   |
|      |        | 14   | ; TRANSMIT A SUPERVISORY FRAME FROM THE PRIMARY STATION WITH THE POLL |
|      |        | 15   | BIT SET AND A RNR COMMAND   |
|      |        | 16   |   |
| 0013 | 747E   | 17   | SEND: MOV A.#7EH ; The SIU receives a flag first                      |
| 0015 | 120066 | 18   | CALL XMITB  |
| 0018 | 748A   | 19   | MOV A.#8AH ; The address is next                                      |
| 001A | 120066 | 20   | CALL XMITB  |
| 001D | 7495   | 21   | MOV A.#093H ; RNR SUP FRAME with P/F=1, NR(P)=4                       |
| 001F | 120066 | 22   | CALL XMITB  |
| 0022 | 747E   | 23   | MOV A.#7EH ; Receive closing flag                                     |
| 0024 | 120066 | 24   | CALL XMITB  |
| 0027 | D2B0   | 25   | SETB P3.0 ; Generate extra SCLK's to                                  |
| 0029 | D2B0   | 26   | SETB P3.0 ; Initiate receive action                                   |
|      |        | 27   |   |
| 002B | E5CB   | 28   | MOV A.#STS ; Check for appropriate status                             |
| 002D | B4722A | 29   | CJNE A.#72H, ERROR  |
|      |        | 30   |   |
|      |        | 31   | ; PREPARE TO RECEIVE RUP1'S RESPONSE TO PRIMARY'S RNR                 |
|      |        | 32   |   |
|      |        | 33   |   |
| 0030 | C2CC   | 34   | RECVR: CLR SI ; Clear SI  |
| 0032 | 7400   | 35   | MOV A.#00H ; Clear ACC  |
| 0034 | 7B0C   | 36   | MOV R3.#12 ; Try 12 times   |
|      |        | 37   |   |
|      |        | 38   |   |
|      |        | 39   | ; LOOK FOR THE OPENING FLAG   |
|      |        | 40   |   |
| 0036 | D2B0   | 41   | WFLAG1: SETB P3.0 ; SCLK  |
| 0038 | A2B1   | 42   | MOV C,P3.1 ; Transmitted data   |
| 003A | 13     | 43   | RRC A   |
| 003B | B47E03 | 44   | CJNE A.#07EH, WFL01   |
| 003E | 020046 | 45   | JMP CONTINU   |
| 0041 | D8F3   | 46   | WFL01: DJNZ R3, WFLAG1  |
| 0043 | 02005A | 47   | JMP ERROR   |
|      |        | 48   |   |
|      |        | 49   |   |
| 0046 | 12005C | 50   | CONTINU: CALL RCVB ; Get SIU's Transmitted address field              |
| 0049 | B48A0E | 51   | CJNE A.#08AH, ERROR   |
| 004C | 12005C | 52   | CALL RCVB   |
| 004F | B4B10B | 53   | CJNE A.#0B1H, ERROR ; Primary expects to receive RR from SIU          |
| 0052 | 12005C | 54   | CALL RCVB   |
| 0055 | B47E02 | 55   | CJNE A.#07EH, ERROR ; Receive closing flag                            |
|      |        | 56   |   |
| 005B | 80FE   | 57   | DONE: JMP DONE  |
|      |        | 58   |   |
| 005A | 80FE   | 59   | ERROR: JMP ERROR  |
|      |        | 60   |   |
|      |        | 61   |   |
| 005C | 7B0B   | 62   | RCVB: MOV R0.#0B ; Initialize the bit counter                         |
| 005E | D2B0   | 63   | GETBIT: SETB P3.0 ; SCLK  |
| 0060 | A2B1   | 64   | MOV C,P3.1 ; Transmitted data   |
| 0062 | 13     | 65   | RRC A   |
| 0063 | D8F9   | 66   | DJNZ R0, GETBIT   |
| 0065 | 22     | 67   | RET   |
|      |        | 68   |   |
|      |        | 69   |   |
|      |        | 70   |   |
| 0066 | 7B09   | 71   | XMITB: MOV R0.#9 ; Initialize the bit counter                         |
| 0068 | 13     | 72   | L3: RRC A ; Put the bit to be transmitted                             |
|      |        | 73   |   |
| 0069 | D801   | 74   | DJNZ R0, L1 ; in the Carry  |
| 006B | 22     | 75   | RET ; When all bits have been sent                                    |
|      |        | 76   |   |
| 006C | 4004   | 77   | L1: JC L2 ; If the carry bit is set, set                              |
|      |        | 78   |   |
| 006E | C2B0   | 79   | CLR P3.0 ; port P3.0 else   |
| 0070 | 80F6   | 80   | JMP L3 ; clear port P3.0  |
|      |        | 81   |   |
| 0072 | D2B0   | 82   | L2: SETB P3.0   |
| 0074 | 80F2   | 83   | JMP L3  |
|      |        | 84   | end   |

Figure 18-17. Loop-Back Mode Software





# CHAPTER 19

## 8044 APPLICATION EXAMPLES

### 19.0 8044 APPLICATIONS EXAMPLES

#### 19.1 INTERFACING THE 8044 TO A MICROPROCESSOR

The 8044 is designed to serve as an intelligent controller for remote peripherals. However, it can also be used as an intelligent HDLC/SDLC front end for a microprocessor, capable of extensively off-loading link control functions for the CPU. In some applications, the 8044 can even be used for communications preprocessing, in addition to data link control.

This section describes a sample hardware interface for attaching the 8044 to an 8088. It is general enough to be extended to other microprocessors such as the 8086 or the 80186.

#### OVERVIEW

A sample interface is shown in Figure 19-1. Transmission occurs when the 8088 loads a 64 byte block of memory with some known data. The 8088 then enables the 8237A to DMA this data to the 8044. When the 8044 has received all of the data from the 8237A, it sends the data in a SDLC frame. The frame is captured by the Spectron Datascope<sup>®</sup> which displays it on a CRT in hex format.

In reception, the Datascope sends an SDLC information frame to the 8044. The 8044 receives the SDLC frame, buffers it, and sends it to the 8088's memory. In this example the 8044 is being operated in the NON-AUTO mode; therefore, it does not need to be polled by a primary station in order to transmit.

#### THE INTERFACE

The 8044 does not have a parallel slave port. The 8044's 32 I/O lines can be configured as a local microprocessor bus master. In this configuration, the 8044 can expand the ROM and RAM memory, control peripherals, and communicate with a microprocessor.

The 8044, like the 8051, does not have a Ready line, so there is no way to put the 8044 in wait state. The clock on the 8044 cannot be stopped. Dual port RAM could still be used, however, software arbitration would be the only way to prevent collisions. Another way to interface the 8044 with another CPU is to put a FIFO or queue between the two processors, and this was the method chosen for this design.

Figure 19-2 shows the schematic of the 8044/8088 interface. It involves two 8 bit tri-state latches, two SR flip-flops, and some logic gates (6 TTL packs). The circuitry implements a one byte FIFO. RS422 transceivers are used, which can be connected to a multidrop link. Figure 19-3 shows the 8088 and support circuitry; the memory and decoders are not shown. It is a basic 8088 Min Mode

system with an 8237A DMA controller and an 8259A interrupt controller.

DMA Channel One transfers a block of memory to the tri-state latch, while Channel Zero transfers a block of data from the latch to 8088's memory. The 8044's Interrupt 0 signal vectors the CPU into a routine which reads from the internal RAM and writes to the latch. The 8044's Interrupt 1 signal causes the chip to read from the latch and write to its on-chip data RAM. Both DMA requests and acknowledges are active low.

Initially, when the power is applied, a reset pulse coming from the 8284A initializes the SR flip-flops. In this initialization state, the 8044's transmit interrupt and the 8088's transmit DMA request are active; however, the software keeps these signals disabled until either of the two processors are ready to transmit. The software leaves the receive signals enabled, unless the receive buffers are full. In this way either the 8088 or the 8044 are always ready to receive, but they must enable the transmit signal when they have prepared a block to transmit. After a block has been transmitted or received, the DMA and interrupt signals return to the initial state.

The receive and transmit buffer sizes for the blocks of data sent between the 8044 and the 8088 have a maximum fixed length. In this case the buffer size was 64 bytes. The buffer size must be less than 192 bytes to enable 8044 to buffer the data in its on-chip RAM. This design allows blocks of data that are less than 64 bytes, and accommodates networks that allow frames of varying size. The first byte transferred between the 8088 and the 8044 is the byte count to follow; thus the 8044 knows how many bytes to receive before it transmits the SDLC frame. However, when the 8044 sends data to the 8088's memory, the 8237A will not know if the 8044 will send less than the count the 8237A was programmed for. To solve this problem, the 8237A is operated in the single mode. The 8044 uses an I/O bit to generate an interrupt request to the 8259A. In the 8088's interrupt routine, the 8237A's receive DMA channel is disabled, thus allowing blocks of data less than 64 bytes to be received.

#### THE SOFTWARE

The software for the 8044 and the 8088 is shown in Table 19-1. The 8088 software was written in PL/M86, and the 8044 software was written in assembly language.

The 8044 software begins by initializing the stack, interrupt priorities, and triggering types for the interrupts. At this point, the SIU parameter registers are initialized. The receive and transmit buffer starting addresses and lengths are loaded for the on-chip DMA. This DMA is for the serial port. The serial station address and the transmit control bytes are loaded too.

\*Datascope is a trademark of Spectron Inc.



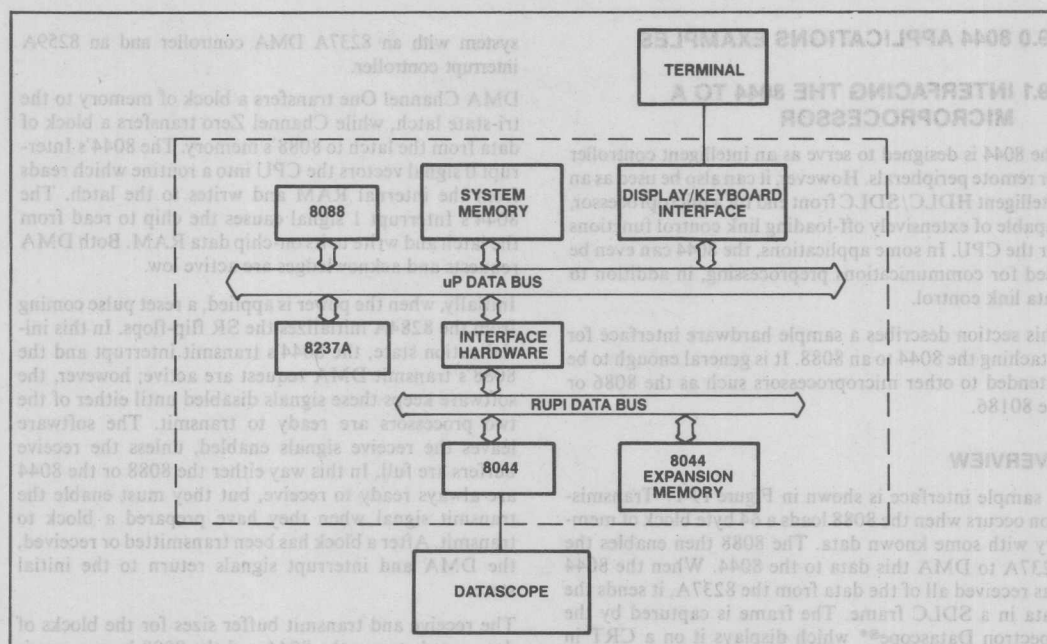


Figure 19-1. Block Diagram of 8088/8044 Interface Test

Once the initialization has taken place, the SIU interrupt is enabled, and the external interrupt which receives bytes from the 8088 is enabled. Setting the 8044's Receive Buffer Empty (RBE) bit enables the receiver. If this bit is reset, no serial data can be received. The 8044 then waits in a loop for either RECEIVE DMA interrupt or the SERIAL INT interrupt.

The RECEIVE DMA interrupt occurs when the 8237A is transferring a block of data to the 8044. The first time this interrupt occurs, the 8044 reads the latch and loads the count value into the R2 register. On subsequent interrupts, the 8044 reads the latch, loads the data into the transmit buffer, and decrements R2. When R2 reaches zero, the interrupt routine sends the data in an SDLC frame, and disables the RECEIVE DMA interrupt. After the frame has been transmitted, a serial interrupt is generated. The SERIAL INT routine detects that a frame has been transmitted and re-enables the RECEIVE DMA interrupt. Thus, while the frame is being transmitted through the SIU, the 8237A is inhibited from sending data to the 8044's transmit buffer.

The TRANSMIT DMA routine sends a block of data from the 8044's receive buffer to the 8088's memory. Normally this interrupt remains disabled. However, if a serial interrupt occurs, and the SERIAL INT routine detects that a frame has been received, it calls the SEND subroutine. The SEND subroutine loads the number of bytes which were received in the frame into the receive buffer. Register R1 points to the receive buffer

and R2 is loaded with the count. The TRANSMIT DMA interrupt is enabled, and immediately upon returning from the SERIAL INT routine, the interrupt is acknowledged. Each time the TRANSMIT DMA interrupt occurs, a byte is read from the receive buffer, written to the latch, and R2 is decremented. When R2 reaches 0, the TRANSMIT DMA interrupt is disabled, the SIU receiver is re-enabled, and the 8044 interrupts the 8088.

The 8088 software simply transmits a block of data and receives a block of data, then stops. The software begins by initializing the 8237A, and the 8259A. It then loads a block of memory with some data and enables the 8237A to transmit the data. In the meantime the 8088 waits in a loop. After a block of data is received from the 8044, the 8088 is interrupted, and it shuts off the 8237A receive DMA.

## CONCLUSION

For the software shown in Table 19-1, the transfer rate from the 8088's memory to the 8044 was measured at 75K bytes/sec. This transfer rate largely depends upon the number of instructions in the 8044's interrupt service routine. Fewer instructions result in a higher transfer rate.

There are many ways of interfacing the 8044 locally to another microprocessor: FIFO's, dual port RAM with software arbitration, and 8255's are just a few. Alternative approaches, which may be more optimal for certain applications, are certainly possible.

**Figure 19-2. 8044 Interface to the 8088**

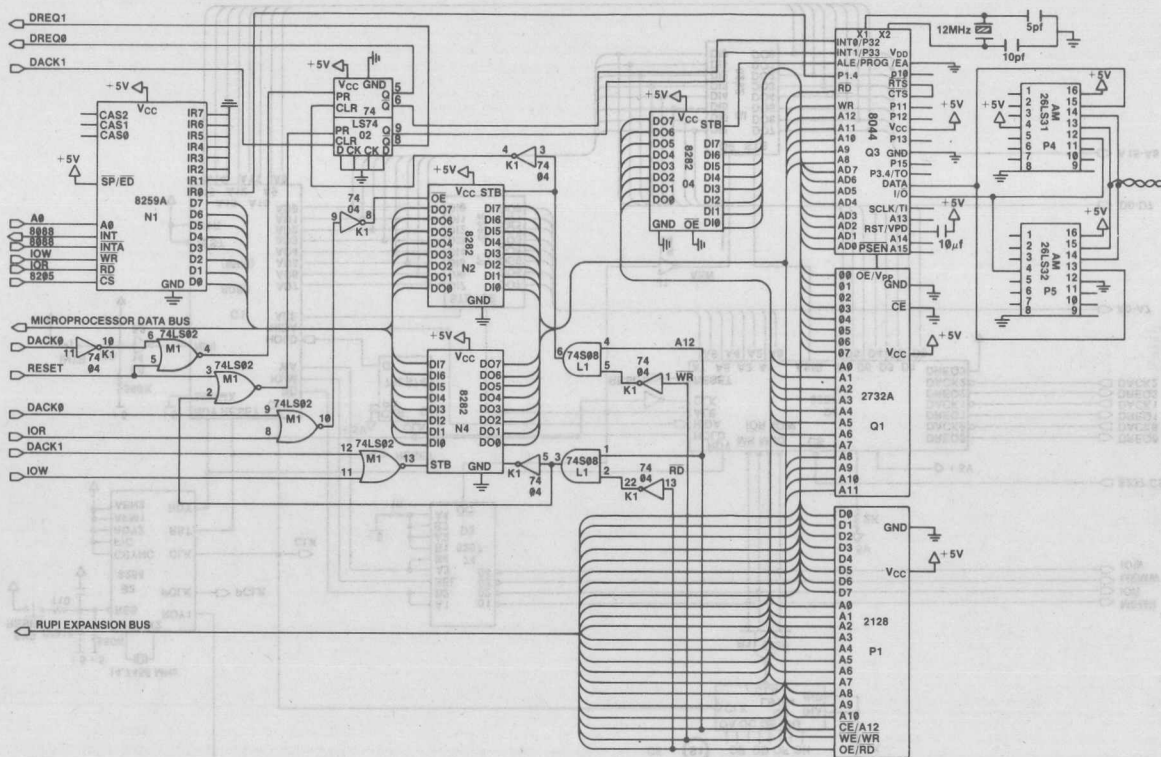


Figure 19-3. 8088 Min Mode System

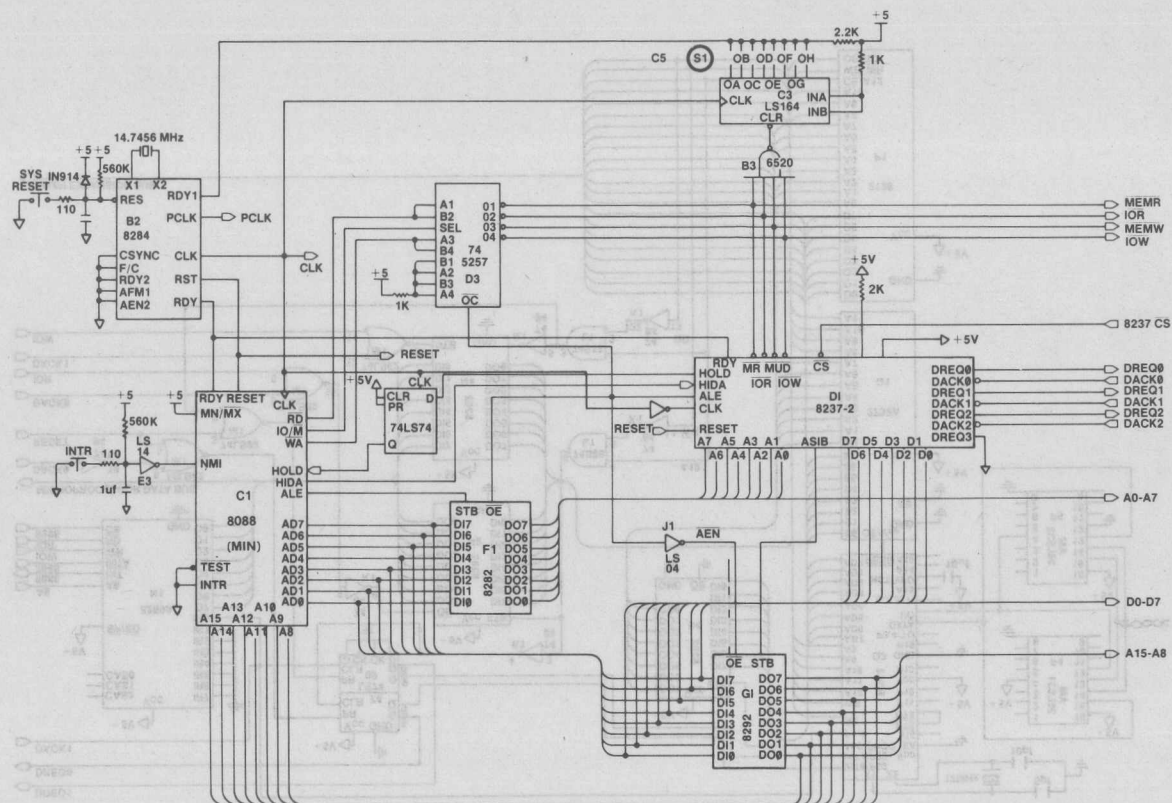


Table 19-1. Transmit and Receive Software for an 8044/8088 System

| LOC         | OBJ | LINE | SOURCE  |
|-------------|-----|------|---|
|             |     | 1    | Sdebug title (8044/8088 INTERFACE)                        |
|             |     | 2    |   |
| 0000        |     | 3    |   |
|             |     | 4    | FIRST_BYTE BIT 0 ; FLAG                                   |
|             |     | 5    |   |
| 0000        |     | 6    | ORG 0   |
| 0000 8024   |     | 7    | SJMP INIT   |
|             |     | 8    |   |
| 0026        |     | 9    | ORG 26H   |
|             |     | 10   |   |
| 0026 7581AA |     | 11   | INIT: MOV SP, #170 ; INITIALIZE STACK                     |
| 0029 75B800 |     | 12   | MOV IP, #00 ; ALL INTERRUPTS ARE EQUAL PRIORITY           |
| 002C 75C954 |     | 13   | MOV SMD, #54H ; TIMER 1 OVERFLOW, NRZI, PRE-FRAME SYNC    |
| 002F 758844 |     | 14   | MOV TCON, #44H ; EDGE TRIGGERED EXTERNAL INTERRUPT 1      |
|             |     | 15   | ; LEVEL TRIGGERED EXTERNAL INTERRUPT 0                    |
|             |     | 16   | ; TIMER 1 ON  |
| 0032 758DEC |     | 17   | MOV TH1, #0ECH ; INITIALIZE TIMER, 3125 BPS               |
| 0035 758920 |     | 18   | MOV TMOD, #20H ; TIMER 1 AUTO RELOAD                      |
|             |     | 19   |   |
| 0038 75DC6A |     | 20   | MOV TBS, #106 ; SET UP SIU PARAMETER REGISTERS            |
| 003B 75DB40 |     | 21   | MOV TBL, #64  |
| 003E 75CC2A |     | 22   | MOV RBS, #42  |
| 0041 75CB40 |     | 23   | MOV RBL, #64  |
| 0044 75CE55 |     | 24   | MOV STAD, #55H  |
| 0047 75DA11 |     | 25   | MOV TCB, #00010001B ; RR, P/F=1                           |
|             |     | 26   |   |
| 004A 901000 |     | 27   | MOV DPTR, #1000H ; DPTR POINTS TO TRI-STATE LATCH         |
| 004D D200   |     | 28   | SETB FIRST_BYTE ; FLAG TO INDICATE FIRST BYTE             |
|             |     | 29   | ; FOR RECEIVE INTERRUPT ROUTINE                           |
| 004F D2CE   |     | 30   | SETB RBE ; READY TO RECEIVE                               |
| 0051 75A894 |     | 31   | MOV IE, #10010100B ; ENABLE RECEIVE DMA AND SIU INTERRUPT |
|             |     | 32   |   |
| 0054 80FE   |     | 33   | SJMP \$ ; WAIT HERE FOR INTERRUPTS                        |
|             |     | 34   |   |
| 0056 80FE   |     | 35   | ERROR: SJMP ERROR   |
|             |     | 36   | +1 \$EJ   |
|             |     | 37   | ***** SUBROUTINES *****                                   |
|             |     | 38   |   |
| 0058 85CD29 |     | 39   | SEND: MOV 41, RFL ; FIRST BYTE IN BLOCK IS COUNT          |
| 005B 7929   |     | 40   | MOV R1, #41 ; POINT TO BLOCK OF DATA                      |
| 005D AACD   |     | 41   | MOV R2, RFL ; LOAD COUNT                                  |
| 005F 0A     |     | 42   | INC R2  |
| 0060 D2A8   |     | 43   | SETB EX0 ; ENABLE DMA TRANSMIT INTERRUPT                  |
| 0062 22     |     | 44   | RET   |
|             |     | 45   |   |
|             |     | 46   |   |
|             |     | 47   |   |
|             |     | 48   | ***** INTERRUPT SERVICE ROUTINES *****                    |
|             |     | 49   |   |
| 0063        |     | 50   | LOC_TMPSET \$ ; SET UP INTERRUPT TABLE JUMP               |
| 0013        |     | 51   | ORG 0013H   |
| 0013 020063 |     | 52   | LJMP RECEIVE_DMA  |
| 0063        |     | 53   | ORG LOC_TMP   |
|             |     | 54   |   |
|             |     | 55   | RECEIVE_DMA:  |



|      |        |     |              |                |   |
|------|--------|-----|--------------|----------------|---|
| 0063 | 10000E | 57  | JBC          | FIRST_BYTE, L1 | ; THE FIRST BYTE TRANSFERRED IS THE COUNT |
| 0066 | E0     | 59  | MOVX         | A, @DPTR       | ; READ THE LATCH                          |
| 0067 | F6     | 60  | MOV          | @R0, A         | ; PUT IT IN TRANSMIT BUFFER               |
| 0068 | 08     | 61  | INC          | R0             |   |
| 0069 | DA08   | 62  | DJNZ         | R2, L2         | ; AFTER READING BYTES,                    |
|      |        | 63  |              |                |   |
| 006B | D2CF   | 64  | SETB         | TBF            | ; SEND DATA                               |
| 006D | D2CD   | 65  | SETB         | RTS            |   |
| 006F | D200   | 66  | SETB         | FIRST_BYTE     |   |
| 0071 | C2AA   | 67  | CLR          | EX1            |   |
|      |        | 68  |              |                |   |
| 0073 | 32     | 69  | L2:          | RETI           |   |
|      |        | 70  |              |                |   |
| 0074 | 786A   | 71  | L1:          | MOV R0, #106   | ; R0 IS A POINTER TO THE TRANSMIT         |
|      |        | 72  |              |                | ; BUFFER STARTING ADDRESS                 |
| 0076 | E0     | 73  | MOVX         | A, @DPTR       | ; PUT THE FIRST BYTE INTO                 |
| 0077 | FA     | 74  | MOV          | R2, A          | ; R2 FOR THE COUNT                        |
| 0078 | 32     | 75  | RETI         |                |   |
|      |        | 76  |              |                |   |
| 0079 |        | 77  | LOC_TMPSET   | \$             |   |
| 0003 |        | 78  | ORG          | 0003H          |   |
| 0003 | 020079 | 79  | LJMP         | TRANSMIT_DMA   |   |
| 0079 |        | 80  | ORG          | LOC_TMP        |   |
|      |        | 81  |              |                |   |
|      |        | 82  | TRANSMIT_DMA |                |   |
|      |        | 83  |              |                |   |
| 0079 | E7     | 84  | MOV          | A, @R1         | ; READ BYTE OUT OF THE RECEIVE BUFFER     |
| 007A | F0     | 85  | MOVX         | @DPTR, A       | ; WRITE IT TO THE LATCH                   |
| 007B | 09     | 86  | INC          | R1             |   |
| 007C | DA08   | 87  | DJNZ         | R2, L3         | ; WHEN ALL BYTES HAVE BEEN SENT           |
|      |        | 88  |              |                |   |
| 007E | C2A8   | 89  | CLR          | IE.0           | ; DISABLE INTERRUPT                       |
| 0080 | C294   | 90  | CLR          | P1.4           | ; CAUSE 8088 INTERRUPT TO TERMINATE DMA   |
| 0082 | D294   | 91  | SETB         | P1.4           |   |
| 0084 | D2CE   | 92  | SETB         | RBE            | ; ENABLE RECEIVER AGAIN                   |
|      |        | 93  |              |                |   |
| 0086 | 32     | 94  | L3:          | RETI           |   |
|      |        | 95  |              |                |   |
|      |        | 96  |              |                |   |
|      |        | 97  |              |                |   |
| 0087 |        | 98  | LOC_TMPSET   | \$             |   |
| 0023 |        | 99  | ORG          | 0023H          |   |
| 0023 | 020087 | 100 | LJMP         | SERIAL_INT     |   |
| 0087 |        | 101 | ORG          | LOC_TMP        |   |
|      |        | 102 |              |                |   |
|      |        | 103 | SERIAL_INT:  |                |   |
|      |        | 104 |              |                |   |
| 0087 | 30CE06 | 105 | JNB          | RBE, RCV       | ; WAS A FRAME RECEIVED                    |
| 008A | 30CF0B | 106 | JNB          | TBF, XMIT      | ; WAS A FRAME TRANSMITTED                 |
| 008D | 020056 | 107 | LJMP         | ERROR          | ; IF NEITHER ERROR                        |
|      |        | 108 |              |                |   |
| 0090 | 20CBC3 | 109 | RCV:         | JB BOV, ERROR  | ; IF BUFFER OVERRUN THEN ERROR            |
| 0093 | 1158   | 110 | CALL         | SEND           | ; SEND THE FRAME TO THE 8088              |
| 0095 | C2CC   | 111 | CLR          | SI             |   |
| 0097 | 32     | 112 | RETI         |                |   |
|      |        | 113 |              |                |   |
| 0098 | C2CC   | 114 | XMIT:        | CLR SI         |   |

```

009A D2AA      115      SETB  EX1
009C 32        116      RETI
                  117
                  118      END
  
```

# SYMBOL TABLE LISTING

| NAME         | TYPE   | VALUE     | ATTRIBUTES |
|--------------|--------|-----------|------------|
| BOV          | B ADDR | 00C8H.3 A |            |
| ERROR        | C ADDR | 0056H A   |            |
| EX0          | B ADDR | 00A8H.0 A |            |
| EX1          | B ADDR | 00A8H.2 A |            |
| FIRST_BYTE   | B ADDR | 0020H.0 A |            |
| IE           | D ADDR | 00A8H A   |            |
| INIT         | C ADDR | 0026H A   |            |
| IP           | D ADDR | 00B8H A   |            |
| L1           | C ADDR | 0074H A   |            |
| L2           | C ADDR | 0073H A   |            |
| L3           | C ADDR | 0086H A   |            |
| LOC.TMP      | C ADDR | 0087H A   |            |
| P1           | D ADDR | 0090H A   |            |
| RBE          | B ADDR | 00C8H.6 A |            |
| RBL          | D ADDR | 00CBH A   |            |
| RBS          | D ADDR | 00CCH A   |            |
| RCV          | C ADDR | 0090H A   |            |
| RECEIVE_DMA  | C ADDR | 0063H A   |            |
| RFL          | D ADDR | 00CDH A   |            |
| RTS          | B ADDR | 00C8H.5 A |            |
| SEND         | C ADDR | 0058H A   |            |
| SERIAL_INT   | C ADDR | 0087H A   |            |
| SI           | B ADDR | 00C8H.4 A |            |
| SMD          | D ADDR | 00C9H A   |            |
| SP           | D ADDR | 0081H A   |            |
| STAD         | D ADDR | 00CEH A   |            |
| TBF          | B ADDR | 00C8H.7 A |            |
| TBL          | D ADDR | 00DBH A   |            |
| TBS          | D ADDR | 00DCH A   |            |
| TCB          | D ADDR | 00DAH A   |            |
| TCON         | D ADDR | 0088H A   |            |
| TH1          | D ADDR | 008DH A   |            |
| TMOD         | D ADDR | 0089H A   |            |
| TRANSMIT_DMA | C ADDR | 0079H A   |            |
| XMIT         | C ADDR | 0098H A   |            |

REGISTER BANK(S) USED: 0, TARGET MACHINE(S): 8044

ASSEMBLY COMPLETE, NO ERRORS FOUND

### Table 19-2. PL/M-86 Compiler Rupi/8088 Interface Example

SERIES-III PL/M-86 V1.0 COMPILATION OF MODULE RUP1\_88  
OBJECT MODULE PLACED IN :F1:R88.OBJ  
COMPILER INVOKED BY: PLM86.86 :F1:R88.SRC

```

$DEBUG
$TITLE ('RUPI/8088 INTERFACE EXAMPLE')
1      RUPI_88.DO;
2      1      DECLARE
                LIT      LITERALLY 'LITERALLY',
                TRUE      LIT      '01H',
                FALSE     LIT      '00H',
                RECV_BUFFER(64)  BYTE,
                XMIT_BUFFER(64)  BYTE,
                I              BYTE,
                WAIT           BYTE,
                /* 8237 PORTS*/
                MASTER_CLEAR_37 LIT      'OFFDDH',
                COMMAND_37      LIT      'OFFDBH',
                ALL_MASK_37      LIT      'OFFDFH',
                SINGLE_MASK_37   LIT      'OFFDAH',
                STATUS_37        LIT      'OFFDBH',
                REQUEST_REQ_37   LIT      'OFFD9H',
                MODE_REQ_37      LIT      'OFFDBH',
                CLEAR_BYTE_PTR_37 LIT      'OFFDCH',
                CH0_ADDR         LIT      'OFFD0H',
                CH0_COUNT        LIT      'OFFD1H',
                CH1_ADDR         LIT      'OFFD2H',
                CH1_COUNT        LIT      'OFFD3H',
                CH2_ADDR         LIT      'OFFD4H',
                CH2_COUNT        LIT      'OFFD5H',
                CH3_ADDR         LIT      'OFFD6H',
                CH3_COUNT        LIT      'OFFD7H',
                /* 8237 BIT ASSIGNMENTS */
                CH0_SEL          LIT      '00H',
                CH1_SEL          LIT      '01H',
                CH2_SEL          LIT      '02H',
                CH3_SEL          LIT      '03H',
                WRITE_XFER       LIT      '04H',
                READ_XFER        LIT      '08H',
                DEMAND_MODE       LIT      '00H',
                SINGLE_MODE       LIT      '40H',
                BLOCK_MODE        LIT      '80H',
                SET_MASK          LIT      '04H',
                $EJECT
                /* 8259 PORTS */
                STATUS_POLL_59   LIT      'OFFE0H',
                ICW1_59           LIT      'OFFE0H',
                OCW1_59           LIT      'OFFE1H',
                OCW2_59           LIT      'OFFE0H',
                OCW3_59           LIT      'OFFE0H',
                ICW2_59           LIT      'OFFE1H',
                ICW3_59           LIT      'OFFE1H',
                ICW4_59           LIT      'OFFE1H',
                /* INTERRUPT SERVICE ROUTINE */
3      1      OFF_RECV_DMA:  PROCEDURE  INTERRUPT 32;
4      2      OUTPUT(SINGLE_MASK_37)=40H;
5      2      WAIT=FALSE;
6      2      END;

```

```

7 1      DISABLE;

/* INITIALIZE 8237 */

8 1      OUTPUT(MASTER_CLEAR_37)    =0;
9 1      OUTPUT(COMMAND_37)         =040H;
10 1     OUTPUT(ALL_MASK_37)        =0FH;
11 1     OUTPUT(MODE_REQ_37)        =(SINGLE_MODE OR WRITE_XFER OR CHO_SEL);
12 1     OUTPUT(MODE_REQ_37)        =(SINGLE_MODE OR READ_XFER OR CH1_SEL);
13 1     OUTPUT(CLEAR_BYTE_PTR_37)  =0;
14 1     OUTPUT(CHO_ADDR)           =00H;
15 1     OUTPUT(CHO_ADDR)           =40H;
16 1     OUTPUT(CHO_COUNT)          =64;
17 1     OUTPUT(CHO_COUNT)          =00;
18 1     OUTPUT(CH1_ADDR)           =40H;
19 1     OUTPUT(CH1_ADDR)           =40H;
20 1     OUTPUT(CH1_COUNT)          =64;
21 1     OUTPUT(CH1_COUNT)          =00;

/* INITIALIZE 8259 */

22 1     OUTPUT(ICW1_59)            =13H; /*SINGLE MODE, EDGE TRIGGERED
23 1     OUTPUT(ICW2_59)            INPUT, 8086 INTERRUPT TYPE*/
24 1     OUTPUT(ICW4_59)            =20H; /*INTERRUPT TYPE 32*/
25 1     OUTPUT(OCW1_59)            =03H; /*AUTO-EOI*/
26 1     *EJECT                     =0FH; /*ENABLE INTERRUPT LEVEL 0*/
27 1     CALL SET$INTERRUPT (32,OFF_RECV_DMA); /*LOAD INTERRUPT VECTOR LOCATION*/

28 1     XMIT_BUFFER(0)=64; /*THE FIRST BYTE IN THE BLOCK OF DATA IS THE NUMBER
29 2     XMIT_BUFFER(I)=I; /* FILL UP THE XMIT_BUFFER WITH DATA */
30 2     END;

31 1     OUTPUT(ALL_MASK_37)=0FH; /*ENABLE CHANNEL 1 AND 2 */

32 1     ENABLE;

33 1     WAIT=TRUE;
34 1     DO WHILE WAIT;
35 2     END; /* A BLOCK OF DATA WILL BE TRANSFERRED TO THE RUP1.
              WHEN THE RUP1 RECEIVES A BLOCK OF DATA IT WILL
              SEND IT TO THE 8088 MEMORY AND INTERRUPT THE 8088.
              THE INTERRUPT SERVICE ROUTINE WILL SHUT OFF THE DMA
              CONTROLLER AND SET 'WAIT' FALSE */

36 1     DO WHILE 1;
37 2     END;

38 1     END;

MODULE INFORMATION:

CODE AREA SIZE      = 00D7H      215D
CONSTANT AREA SIZE  = 0000H      0D
VARIABLE AREA SIZE  = 0082H     130D
MAXIMUM STACK SIZE  = 001EH      30D
124 LINES READ
0 PROGRAM WARNINGS
0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION

```



# A HIGH PERFORMANCE NETWORK USING THE 8044

## 19.2.1 Introduction

This section describes the design of an SDLC data link using the 8044 (RUP1) to implement a primary station and a secondary station. The design was implemented and tested. The following discussion assumes that the reader understands the 8044 and SDLC. This section is divided into two parts. First the data link design example is discussed. Second the software modules used to implement the data link are described. To help the reader understand the discussion of the software, flow charts and software listings are displayed in Appendix A and Appendix B, respectively.

### Application Description

This particular data link design example uses a two wire half-duplex multidrop topology as shown in figure 19-4. In an SDLC multidrop topology the primary station communicates with each secondary station. The secondary stations communicate only to the primary. Because of this hierarchical architecture, the logical topology for an SDLC multidrop is a star as shown in figure 19-5. Although the physical topology of this data link is multidrop, the easiest way to understand the information flow is to think of the logical (star) topology. The term data link in this case refers to the logical communication pathways between the primary station and the secondary stations. The data links are shown in figure 19-5 as two way arrows.

The application example uses dumb async terminals to interface to the SDLC network. Each secondary station has an async terminal connected to it. The secondary stations are in effect protocol converters which allows any async terminal to communicate with any other async terminal on the network. The secondary stations use an 8044 with a UART to convert SDLC to async. Figure 19-6 displays a block diagram of the data link. The primary station, controls the data link. In addition to data link control the primary provides a higher level layer which is a path control function or networking layer. The primary serves as a message exchange or switch. It receives information from one secondary station and retransmits it to another secondary station. Thus a virtual end to end connection is made between any two secondary stations on the network.

Three separate software modules were written for this network. The first module is a Secondary Station Driver (SSD) which provides an SDLC data link interface and a user interface. This module is a general purpose driver which requires application software to run it. The user interface to the driver provides four functions: OPEN,

CLOSE, TRANSMIT, and SIU\_RECV. Using these four functions properly will allow any application software to communicate over this SDLC data link without knowing the details of SDLC. The secondary station driver uses the 8044's AUTO mode.

The second module is an example of application software which is linked to the secondary station driver. This module drives the 8251A, buffers data, and interfaces with the secondary station driver's user interface.

The third module is a primary station, which is a stand-alone program (i.e., it is not linked to any other module). The primary station uses the 8044's NON-AUTO or FLEXIBLE mode. In addition to controlling the data link it acts as a message switch. Each time a secondary station transmits a frame, it places the destination address of the frame in the first byte of the information or I field. When the primary station receives a frame, it removes the first byte in the I field and retransmits the frame to the secondary station whose address matches this byte.

This network provides two complete layers of the OSI (Open Systems Interconnection) reference model: the physical layer and the data link layer. The physical layer implementation uses the RS-422 electrical interface. The mechanical medium consists of ribbon cable and connectors. The data link layer is defined by SDLC. SDLC's use of acknowledgements and frame numbering guarantees that messages will be received in the same order in which they were sent. It also guarantees message integrity over the data link. However this network will not guarantee secondary to secondary message delivery, since there are acknowledgements between secondary stations.

## 19.2.2 Hardware

The schematic of the hardware is given in figure 19-7. The 8251A is used as an async communications controller, in support of the 8044. TxRDY and RxRDY on the 8251A are both tied to the two available external interrupts of the 8044 since the secondary station driver is totally interrupt driven. The 8044 buffers the data and some variables in a 2016 (2K x 8 static RAM). The 8254 programmable interval timer is employed as a programmable baud rate generator and system clock driver for the 8251A. The third output from the 8254 could be used as an external baud rate generator for the 8044. The 2732A shown in the diagram was not used since the software for both the primary and secondary stations used far less than the 4 Kbytes provided on the 8744. For the async interface, the standard RS-232

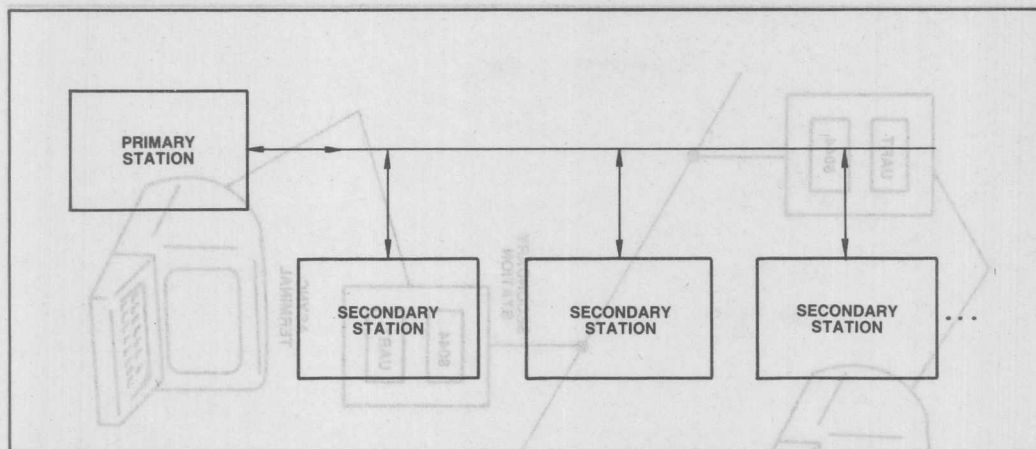


Figure 19-4. SDLC Multidrop Topology

mechanical and electrical interface was used. For the SDLC channel, a standard two wire three state RS-422 driver is used. A DIP switch connected to one of the available ports on the 8044 allows the baud rate, parity, and stop bits to be changed on the async interface. The primary station hardware does not use the USART, 8254, nor the RS-232 drivers.

### 19.2.3 SDLC Basic Repertoire

The SDLC commands and responses implemented in the data link include the SDLC Basic Repertoire as defined in the IBM SDLC General Information manual. Table 19-3 shows the commands and responses that the primary and the secondary station in this data link design recognize and send.

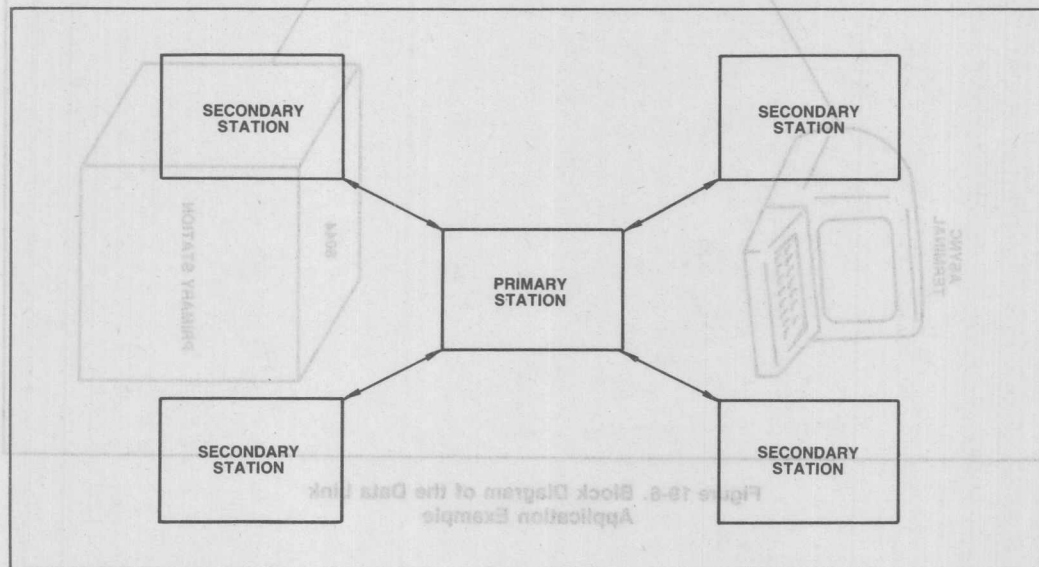


Figure 19-5. SDLC Logical Topology

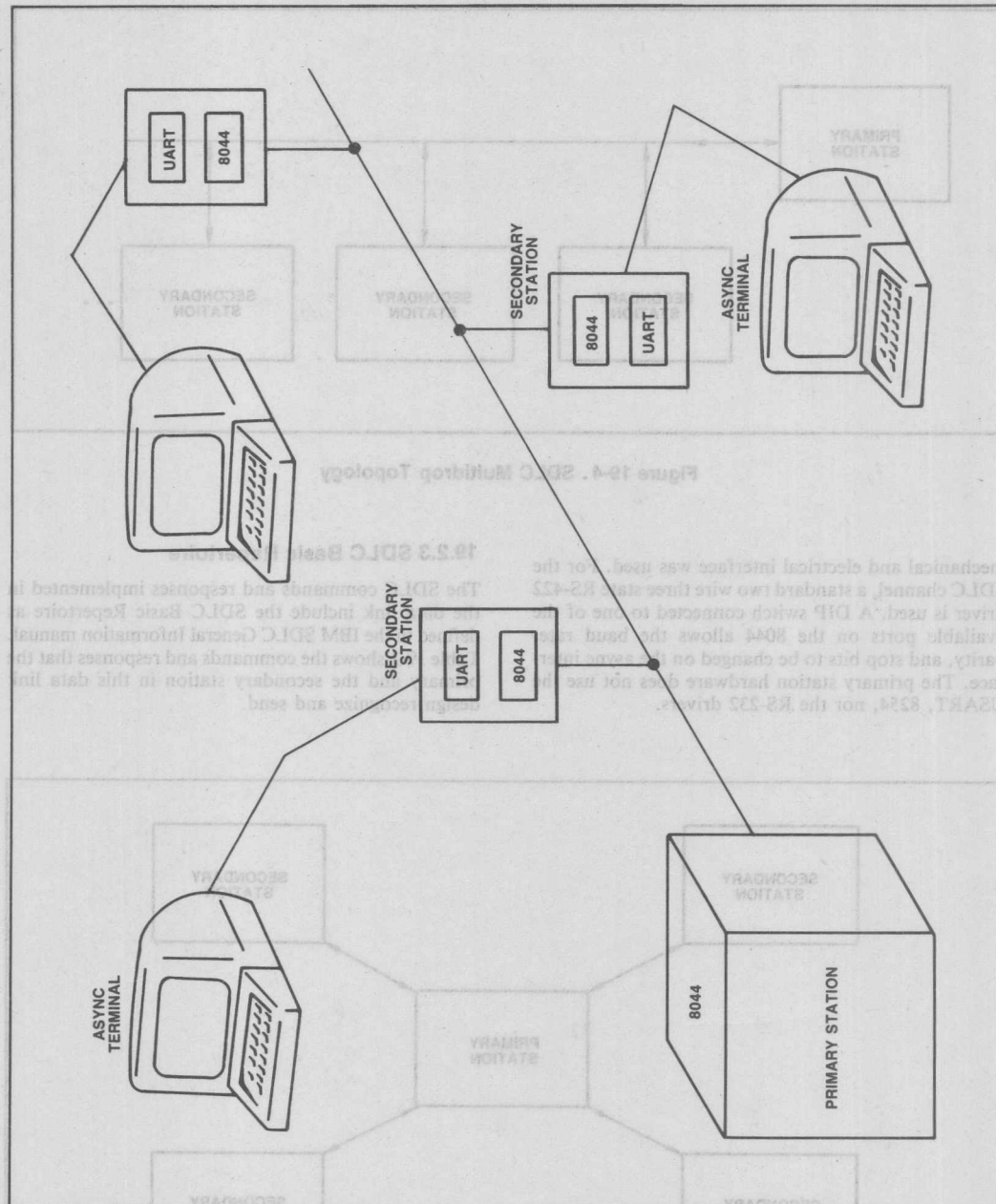


Figure 19-6. Block Diagram of the Data Link Application Example

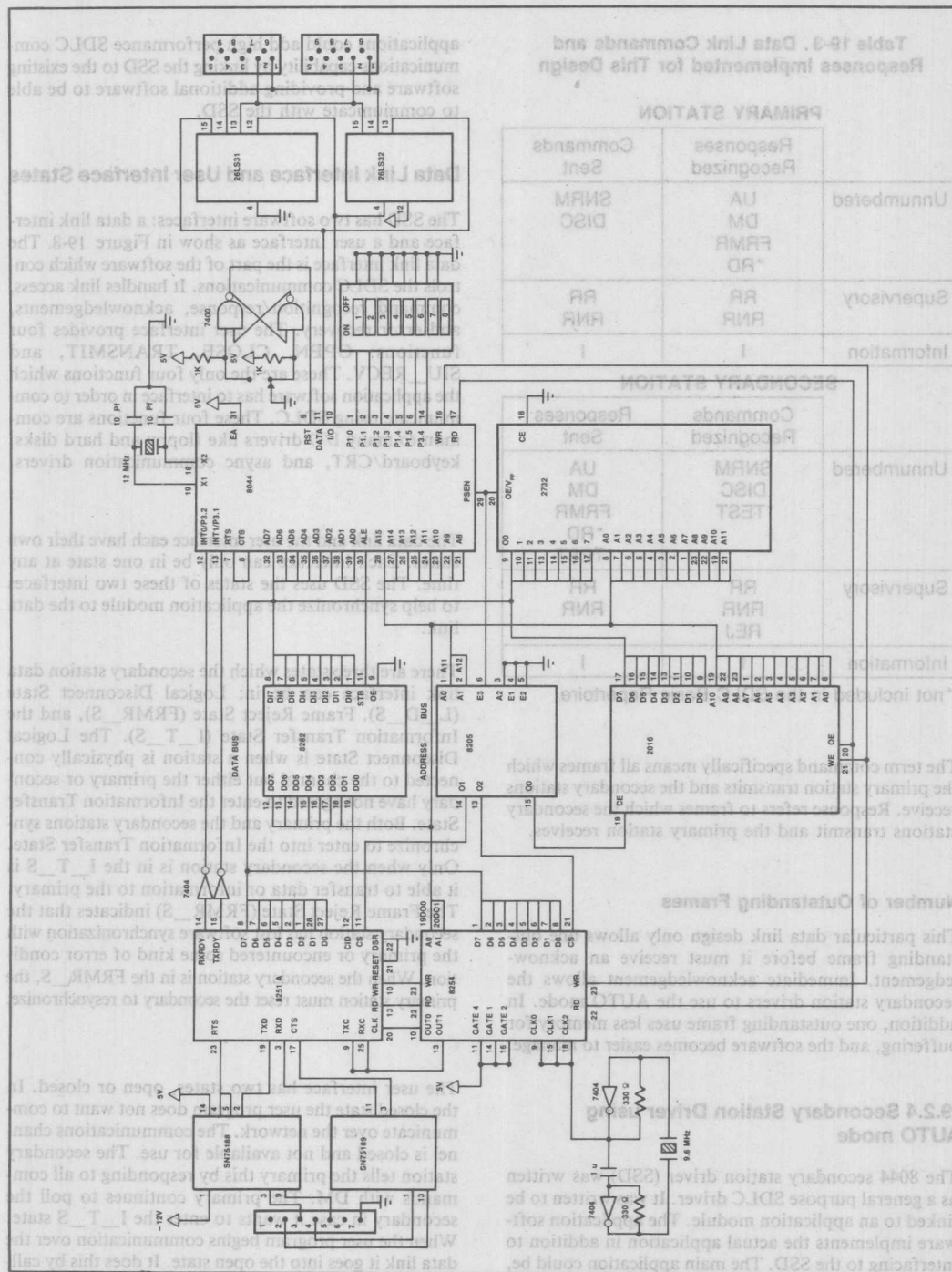
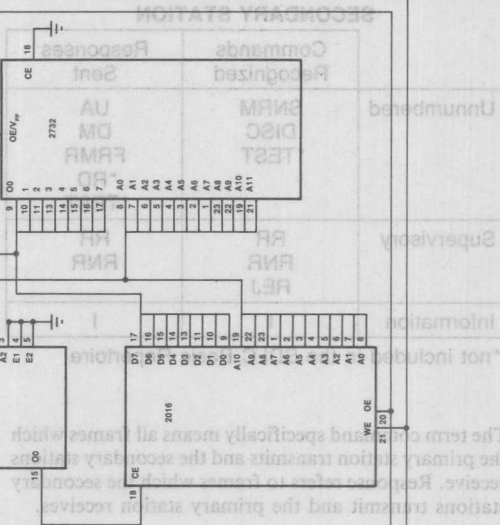


Figure 19-7. Schematic of Async/SDLC Secondary Station Protocol Converter

Table 19-3. Data Link Commands and Responses Implemented for This Design

| Unnumbered    | Supervisory | Information |
|---------------|-------------|-------------|
| UA            | RR          | 1           |
| DM            | RNR         |             |
| FRMR          |             |             |
| *RD           |             |             |
| DISC          |             |             |
| SNRM          |             |             |
| Commands Sent |             |             |



Number of Outstanding Frames

This particular data link design only allows standing frame before it must receive an acknowledgment. Immediate acknowledgment is required. In addition, one outstanding frame uses less buffering, and the software becomes simpler to write.

### 19.2.4 Secondary Station Driver

The 8044 secondary station driver (SSD) is written as a general purpose SDLC driver. It is linked to an application module. The application software implements the actual application in addition to interfacing to the SSD. The main application could be a printer or plotter, a medical instrument, or a terminal. The SSD is the SDLC communications. Existing 8021



## Responses Implemented for This Design

### PRIMARY STATION

|             | Responses Recognized    | Commands Sent |
|-------------|-------------------------|---------------|
| Unnumbered  | UA<br>DM<br>FRMR<br>*RD | SNRM<br>DISC  |
| Supervisory | RR<br>RNR               | RR<br>RNR     |
| Information | I                       | I             |

### SECONDARY STATION

|             | Commands Recognized   | Responses Sent                   |
|-------------|-----------------------|----------------------------------|
| Unnumbered  | SNRM<br>DISC<br>*TEST | UA<br>DM<br>FRMR<br>*RD<br>*TEST |
| Supervisory | RR<br>RNR<br>REJ      | RR<br>RNR                        |
| Information | I                     | I                                |

\*not included in the SDLC Basic Repertoire

The term command specifically means all frames which the primary station transmits and the secondary stations receive. Response refers to frames which the secondary stations transmit and the primary station receives.

### Number of Outstanding Frames

This particular data link design only allows one outstanding frame before it must receive an acknowledgement. Immediate acknowledgement allows the secondary station drivers to use the AUTO mode. In addition, one outstanding frame uses less memory for buffering, and the software becomes easier to manage.

### 19.2.4 Secondary Station Driver using AUTO mode

The 8044 secondary station driver (SSD) was written as a general purpose SDLC driver. It was written to be linked to an application module. The application software implements the actual application in addition to interfacing to the SSD. The main application could be, a printer or plotter, a medical instrument, or a terminal. The SSD is independent of the main application, it just provides the SDLC communications. Existing 8051

applications could add high performance SDLC communications capability by linking the SSD to the existing software and providing additional software to be able to communicate with the SSD.

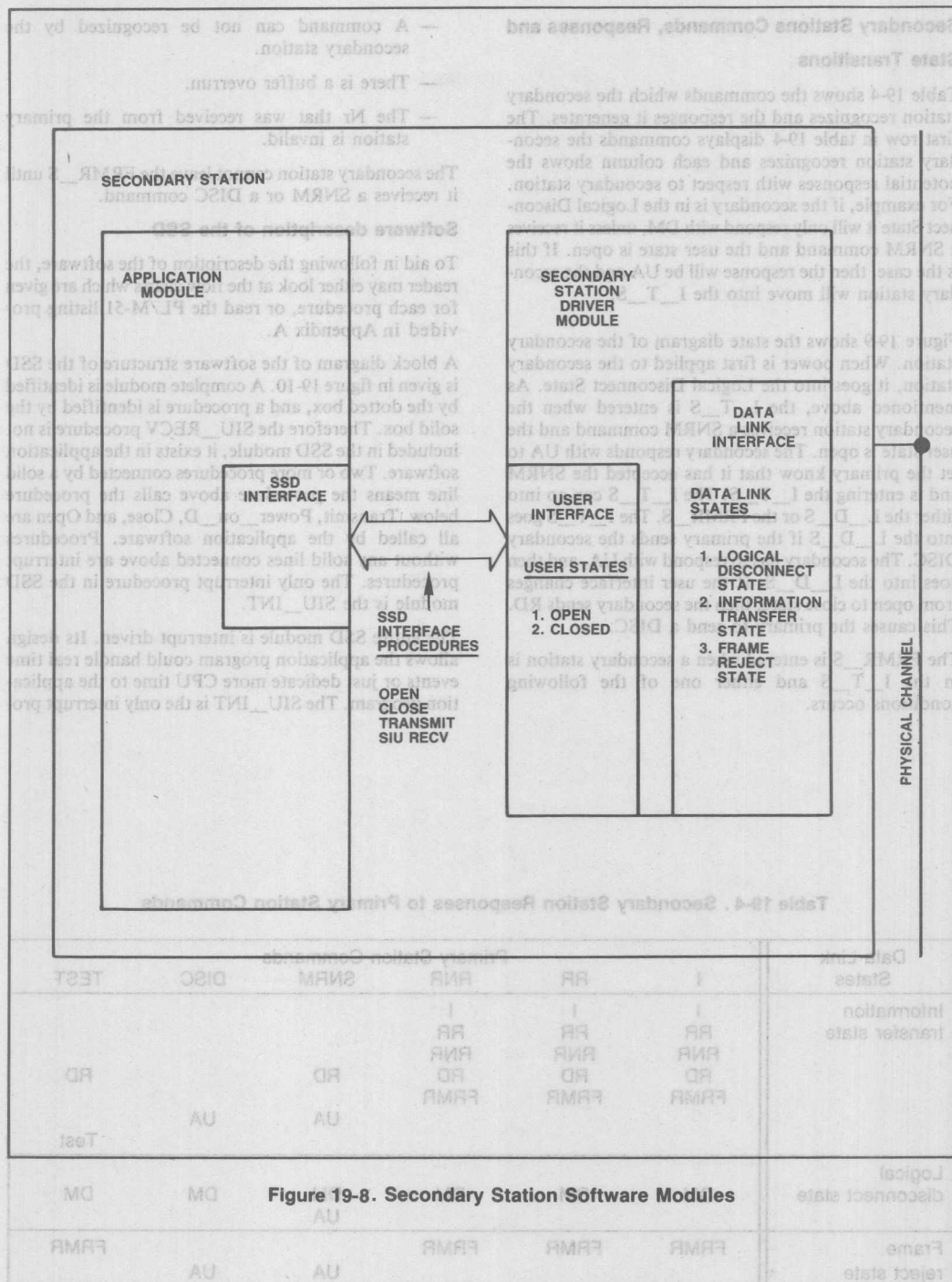
### Data Link Interface and User Interface States

The SSD has two software interfaces: a data link interface and a user interface as show in Figure 19-8. The data link interface is the part of the software which controls the SDLC communications. It handles link access, command recognition/response, acknowledgements, and error recovery. The user interface provides four functions: OPEN, CLOSE, TRANSMIT, and SIU\_RECV. These are the only four functions which the application software has to interface in order to communicate using SDLC. These four functions are common to many I/O drivers like floppy and hard disks, keyboard/CRT, and async communication drivers.

The data link and the user interface each have their own states. Each interface can only be in one state at any time. The SSD uses the states of these two interfaces to help synchronize the application module to the data link.

There are three states which the secondary station data link interface can be in: Logical Disconnect State (L\_D\_S). Frame Reject State (FRMR\_S), and the Information Transfer State (I\_T\_S). The Logical Disconnect State is when a station is physically connected to the channel but either the primary or secondary have not agreed to enter the Information Transfer State. Both the primary and the secondary stations synchronize to enter into the Information Transfer State. Only when the secondary station is in the I\_T\_S is it able to transfer data or information to the primary. The Frame Reject State (FRMR\_S) indicates that the secondary station has lost software synchronization with the primary or encountered some kind of error condition. When the secondary station is in the FRMR\_S, the primary station must reset the secondary to resynchronize.

The user interface has two states, open or closed. In the closed state the user program does not want to communicate over the network. The communications channel is closed and not available for use. The secondary station tells the primary this by responding to all commands with DM. The primary continues to poll the secondary in case it wants to enter the I\_T\_S state. When the user program begins communication over the data link it goes into the open state. It does this by calling the OPEN procedure. When the user interface is in the open state it may transfer information to the primary.



## Secondary Stations Commands, Responses and State Transitions

Table 19-4 shows the commands which the secondary station recognizes and the responses it generates. The first row in table 19-4 displays commands the secondary station recognizes and each column shows the potential responses with respect to secondary station. For example, if the secondary is in the Logical Disconnect State it will only respond with DM, unless it receives a SNRM command and the user state is open. If this is the case, then the response will be UA and the secondary station will move into the I\_T\_S.

Figure 19-9 shows the state diagram of the secondary station. When power is first applied to the secondary station, it goes into the Logical Disconnect State. As mentioned above, the I\_T\_S is entered when the secondary station receives a SNRM command and the user state is open. The secondary responds with UA to let the primary know that it has accepted the SNRM and is entering the I\_T\_S. The I\_T\_S can go into either the L\_D\_S or the FRMR\_S. The I\_T\_S goes into the L\_D\_S if the primary sends the secondary DISC. The secondary has to respond with UA, and then goes into the L\_D\_S. If the user interface changes from open to close state, then the secondary sends RD. This causes the primary to send a DISC.

The FRMR\_S is entered when a secondary station is in the I\_T\_S and either one of the following conditions occurs.

- A command can not be recognized by the secondary station.
- There is a buffer overrun.
- The Nr that was received from the primary station is invalid.

The secondary station cannot leave the FRMR\_S until it receives a SNRM or a DISC command.

### Software description of the SSD

To aid in following the description of the software, the reader may either look at the flow charts which are given for each procedure, or read the PL/M-51 listing provided in Appendix A.

A block diagram of the software structure of the SSD is given in figure 19-10. A complete module is identified by the dotted box, and a procedure is identified by the solid box. Therefore the SIU\_RECV procedure is not included in the SSD module, it exists in the application software. Two or more procedures connected by a solid line means the procedure above calls the procedure below. Transmit, Power\_on\_D, Close, and Open are all called by the application software. Procedures without any solid lines connected above are interrupt procedures. The only interrupt procedure in the SSD module is the SIU\_INT.

The entire SSD module is interrupt driven. Its design allows the application program could handle real time events or just dedicate more CPU time to the application program. The SIU\_INT is the only interrupt pro-

Table 19-4. Secondary Station Responses to Primary Station Commands

| Data Link States           | Primary Station Commands |      |      |      |      |      |
|----------------------------|--------------------------|------|------|------|------|------|
|                            | I                        | RR   | RNR  | SNRM | DISC | TEST |
| Information transfer state | I                        | I    | I    |      |      |      |
|                            | RR                       | RR   | RR   |      |      |      |
|                            | RNR                      | RNR  | RNR  |      |      |      |
|                            | RD                       | RD   | RD   | RD   |      | RD   |
|                            | FRMR                     | FRMR | FRMR | UA   | UA   | Test |
| Logical disconnect state   | DM                       | DM   | DM   | DM   | DM   | DM   |
|                            |                          |      |      | UA   |      |      |
| Frame reject state         | FRMR                     | FRMR | FRMR |      |      | FRMR |
|                            |                          |      |      | UA   | UA   |      |

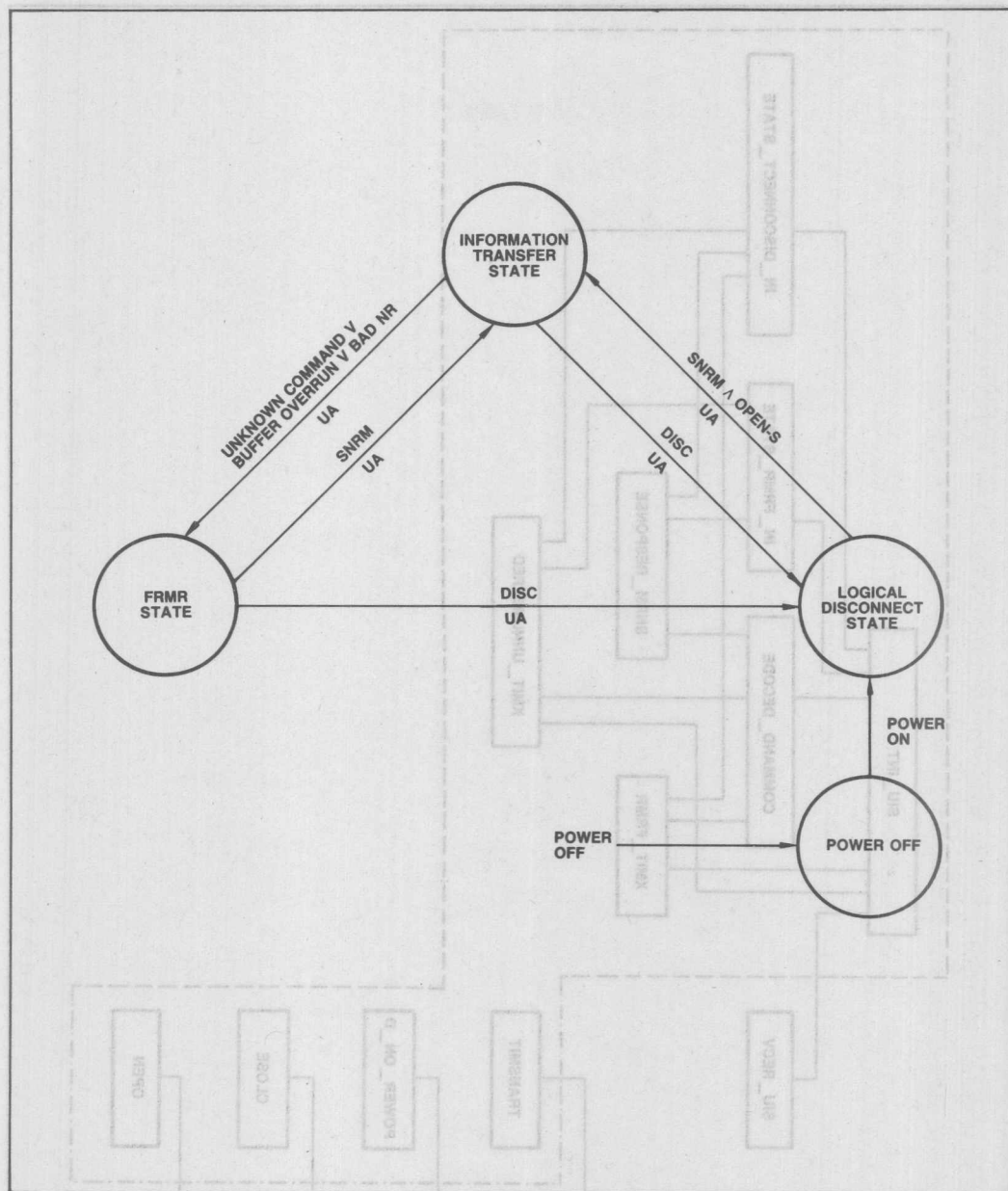


Figure 19-9. State Diagram of Secondary Station



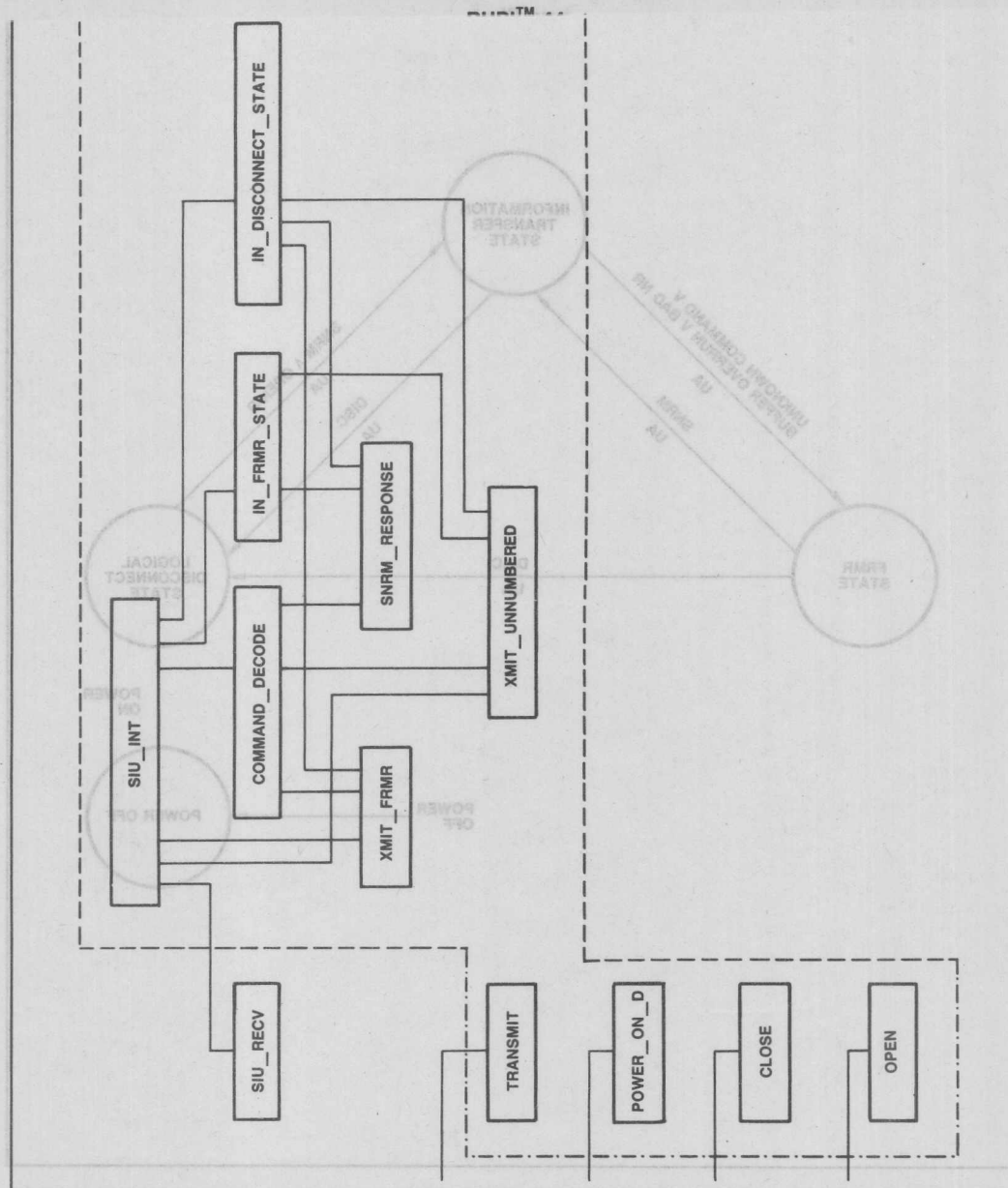


Figure 19-10. Secondary Station Driver

cedure in the SSD. It is automatically entered when an SIU interrupt occurs. This particular interrupt can be the lowest priority interrupt in the system.

### SSD Initialization

Upon reset the application software is entered first. The application software initializes its own variables then calls Power\_On\_D which is the SSD's initialization routine. The SSD's initialization sets up the transmit and receive data buffer pointers (TBS and RBS), the receive buffer length (RBL), and loads the State variables. The STATION\_STATE begins in the L\_D\_S state, and the USER\_STATE begins in the closed state. Finally Power\_On\_D initializes XMIT\_BUFFER\_EMPTY which is a bit flag. This flag serves as a semaphore between the SSD and the application software to indicate the status of the on chip transmit buffer. The SSD does not set the station address. It is the application software's responsibility to do this. After initialization, the SSD is ready to respond to all of the primary stations commands. Each time a frame is received with a matching station address and a good CRC, the SIU\_INT procedure is entered.

### SIU\_INT Procedure

The first thing the SIU\_INT procedure clears the serial interrupt bit (SI) in the STS register. If the SIU\_INT procedure returns with this bit set, another SI interrupt will occur.

The SIU\_INT procedure is branches three independent cases. The first case is entered if the STATION\_STATE is not in the I\_T\_S. If this is true, then the SIU is not in the AUTO mode, and the CPU will have to respond to the primary on its own. (Remember that the AUTO mode is entered when the STATION\_STATE enters into I\_T\_S.) If the STATION\_STATE is in the I\_T\_S, then either the SIU has just left the AUTO mode, or is still in the AUTO mode. This is the second and third case respectively.

In the first case, if the STATION\_STATE is not in the I\_T\_S, then it must be in either the L\_D\_S or the FRMR\_S. In either case a separate procedure is called based on which state the station is in. The In\_Disconnect\_State procedure sends to the primary a DM response, unless it received a SNRM command and the USER\_STATE equals open. In that case the SIU sends an UA and enters into the I\_T\_S. The In\_FRMR\_State procedure will send the primary the FRMR response unless it received either a DISC or an SNRM. If the primary's command was a DISC, then the secondary will send an UA and enter into the L\_D\_S. If the primary's command was a SNRM, then the secondary will send an UA, enter into the I\_T\_S, and clear NSNR register.

For the second case, if the STATION\_STATE is in the I\_T\_S but the SIU left the AUTO mode, then the CPU must determine why the AUTO mode was exited, and generate a response to the primary. There are four reasons for the SIU to automatically leave the AUTO mode. The following is a list of these reasons, and the responses given by the SSD based on each reason.

1. The SIU has received a command field it does not recognize.

Response: If the CPU recognizes the command, it generates the appropriate response. If neither the SIU nor the CPU recognize the command, then a FRMR response is sent.

2. The SIU has received a Sequence Error Sent (SES=1 in NSNR register).  $Nr(P) \neq Ns(S)+1$ , and  $Nr(P) \neq Ns(S)$ .

Response: Send FRMR.

3. A buffer overrun has occurred. BOV=1 in STS register.

Response: Send FRMR.

4. An I frame with data was received while RPB=1.

Response: Go back into AUTO mode and send an AUTO mode response.

In addition to the above reasons, there is one condition where the CPU forces the SIU out of the AUTO mode. This is discussed in the SSD's User Interface Procedures section in the CLOSED procedure description.

Finally, case three is when the STATION\_STATE is in the I\_T\_S and the AUTO mode. The CPU first looks at the TBF bit. If this bit is 0 then the interrupt may have been caused by a frame which was transmitted and acknowledged. Therefore the XMIT\_BUFFER\_EMPTY flag is set again indicating that the application software can transmit another frame.

The other reason this section of code could be entered is if a valid I frame was received. When a good I frame is received the RBE bit equals 0. This means that the receiver is disabled. If the primary were to poll the 8044 while RBE=0, it would time out since no response would be given. Time outs reduce network throughput. To improve network performance, the CPU first sets RBP, then sets RBE. Now when the primary polls the 8044 an immediate RNR response is given. At this point the SSD calls the application software procedure SIU\_RECV and passes the length of the data as a parameter. The SIU\_RECV procedure reads the data out of the receive buffer then returns to the SSD module. Now that the receive information has been transferred, RBP can be cleared.

### Command\_Decode Procedure

The Command\_Decode procedure is called from the SIU\_INT procedure when the STATION\_STATE = I\_T\_S and the SIU left the AUTO mode as a result

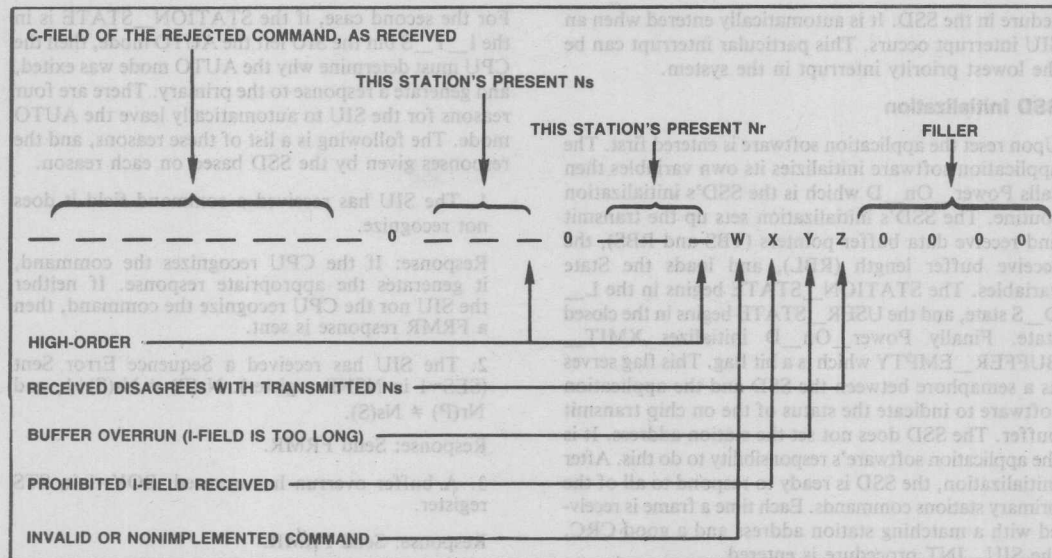


Figure 19-11. Information Field of the FRMR Response, as Transmitted

of not being able to recognize the receive control byte. Commands which the SIU AUTO mode does not recognize are handled here. The commands recognized in this procedure are: SNRM, DISC, and TEST. Any other command received will generate a Frame Reject with the nonimplemented command bit set in the third data byte of the FRMR frame. Any additional unnumbered frame commands which the secondary station is going to implement, should be implemented in this procedure.

If a SNRM is received the command\_decode procedure calls the SNRM\_Response procedure. The SNRM\_Response procedure sets the STATION\_STATE = I\_T\_S, clears the NSNR register and responds with an UA frame. If a DISC is received, the command\_decode procedure sets the STATION\_STATE = L\_D\_S, and responds with an UA frame. When a TEST frame is received, and there is no buffer overrun, the command\_decode procedure responds with a TEST frame retransmitting the same data it received. However if a TEST frame is received and there is a buffer overrun, then a TEST frame will be sent without any data, instead of a FRMR with the buffer overrun bit set.

#### Frame Reject Procedures

There are two procedures which handle the FRMR state: XMIT\_FRMR and IN\_FRMR\_STATE. XMIT\_FRMR is entered when the secondary station first goes into the FRMR state. The frame reject response frame contains the FRMR response in the command field plus three additional data bytes in the I field. Figure 19-11 displays the format for the three data byte in the I field

of a FRMR response. The XMIT\_FRMR procedure sets up the Frame Reject response frame based on the parameter REASON which is passed to it. Each place in the SSD code that calls the XMIT\_FRMR procedure, passes the REASON that this procedure was called, which in turn is communicated to the primary station. The XMIT\_FRMR procedure uses three bytes of internal RAM which it initializes for the correct response. The TBS and TBL registers are then changed to point to the FRMR buffer so that when a response is sent these three bytes will be included in the I field.

The IN\_FRMR\_STATE procedure is called by the SIU\_INT procedure when the STATION\_STATE already is in the FRMR state and a response is required. The IN\_FRMR\_STATE procedure will only allow two commands to remove the secondary station from the FRMR state: SNRM and DISC. Any other command which is received while in the FRMR state will result a FRMR response frame.

#### XMIT\_UNNUMBERED Procedure

This is a general purpose transmit procedure, used only in the FLEXIBLE mode, which sends unnumbered responses to the primary. It accepts the control byte as a parameter, and also expects the TBL register to be set before the procedure is called. This procedure waits until the frame has been transmitted before returning. If this procedure returned before the transmit interrupt was generated, the SIU\_INT routine would be entered. The SIU\_INT routine would not be able to distinguish this condition.

SSD's User Interface Procedures -- OPEN, CLOSE, TRANSMIT, SIU\_RECV -- are discussed in the following section.

The OPEN procedure is the simplest of all, it changes the USER\_STATE to OPEN\_S then returns. This lets the SSD know that the user wants to open the channel for communications. When the SSD receives a SNRM command, it checks the USER\_STATE. If the USER\_STATE is open, then the SSD will respond with an UA, and the STATION\_STATE enters the I\_T\_S.

The CLOSE procedure is also simple, it changes the USER\_STATE to CLOSED\_S and sets the AM bit to 0. Note that when the CPU sets the AM bit to 0 it puts the SIU out of the AUTO mode. This event is asynchronous to the events on the network. As a result an I frame can be lost. This is what can happen.

1. AM is set to 0 by the CLOSE Procedure.
2. An I frame is received and a SI interrupt occurs.
3. The SIU\_INT procedure enters case 2. (STATION\_STATE = I\_T\_S, and AM = 0)
4. Case 2 detects that the USER\_STATE = CLOSED\_S, sends a RD response and ignores the fact that an I frame was received.

Therefore it is advised to never call the CLOSE procedure or take the SIU out of the AUTO mode when it is receiving I frames or an I frame will be lost.

For both the TRANSMIT and SIU\_RECV procedures, it is the application software's job to put data into the transmit buffer, and take data out of the receive buffer. The SSD does not transfer data in or out of its transmit or receive buffers because it does not know what kind of buffering the application software is implementing. What the SSD does do is notify the application software when the transmit buffer is empty, XMIT\_BUFFER\_EMPTY = 1, and when the receive buffer is full.

One of the functions that the SSD performs to synchronize the application software to the SDLC data link. However some of the synchronization must also be done by the application software. Remember that the SSD does not want to hang up the application software waiting for some event to occur on the SDLC data link, therefore the SSD always returns to the application software as soon as possible.

For example, when the application software calls the OPEN procedure, the SSD returns immediately. The application software thinks that the SDLC channel is now open and it can transmit. This is not the case. For the channel to be open, the SSD must receive a SNRM from the primary and respond with a UA. However, the SSD does not want to hang up the application software waiting for a SNRM from the primary before returning from the OPEN procedure. When the

TRANSMIT procedure is called, the SSD expects the STATION\_STATE to be in the I\_T\_S. If it isn't, the SSD refuses to transmit the data. The TRANSMIT procedure first checks to see if the USER\_STATE is open, if not the USER\_STATE\_CLOSED parameter is passed back to the application module. The next thing TRANSMIT checks is the STATION\_STATE. If this is not open, then TRANSMIT passes back LINK\_DISCONNECTED. This means that the USER\_STATE is open, but the SSD hasn't received a SNRM command from the primary yet. Therefore, the application software should wait awhile and try again. Based on network performance, one knows the maximum amount of time it will take for a station to be polled. If the application software waits this length of time and tries again but still gets a LINK\_DISCONNECTED parameter passed back, higher level recovery must be implemented.

Before loading the transmit buffer and calling the TRANSMIT procedure, the application software must check to see that XMIT\_BUFFER\_EMPTY = 1. This flag tells the application software that it can write new data into the transmit buffer and call the TRANSMIT procedure. After the application software has verified that XMIT\_BUFFER\_EMPTY = 1, it fills the transmit buffer with the data and calls the TRANSMIT procedure passing the length of the buffer as a parameter. The TRANSMIT procedure checks for three reasons why it might not be able to transmit the frame. If any of these three reasons are true, the TRANSMIT procedure returns a parameter explaining why it couldn't send the frame. If the application software receives one of these responses, it must rectify the problem and try again. Assuming these three conditions are false, then the SSD clears XMIT\_BUFFER\_EMPTY, attempts to send the data and returns the parameter DATA\_TRANSMITTED. XMIT\_BUFFER\_EMPTY will not be set to 1 again until the data has been transmitted and acknowledged.

The SIU\_RECV procedure must be incorporated into the application software module. When a valid I frame is received by the SIU, it calls the SIU\_RECV procedure and passes the length of the received data as a parameter. The SIU\_RECV procedure must remove all of the data from the receive buffer before returning to the SIU\_INT procedure.

## Linking up to the SSD

Figure 19-12 shows necessary parts to include in a PL/M-51 application program that will be linked to the SSD module. RL51 is used to link and locate the SSD and application modules. The command line used to do this is:



```
RL51 SSD.obj,filename.obj,PLM51.LIB TO filename
& RAMSIZE(192)
```

```
$registerbank(0)
user$mod: do;
$include (reg44.dcl)
declare
  lit      literally 'literally',
  buffer_length  lit      '60',
  siu_xmit_buffer  byte    external idata,
  (buffer_length)
  siu_rcv_buffer  byte    external,
  (buffer_length)
  xmit_buffer_empty  bit    external;
/* external procedures */

power_on_d: procedure external;
end power_on_d;

close: procedure external using 1;
end close;

open: procedure external using 1;
end open;

transmit: procedure
  (xmit_buffer_length) byte external;
  declare xmit_buffer_length byte;
end transmit;

/* local procedures */

siu_rcv: procedure (length) public using 1;
  declare length byte;
end siu_rcv;
```

**Figure 19-12. Applications Module Link Information**

### PL/M-51 and Register Banks

The 8044 has four register banks. PL/M-51 assumes that an interrupt procedure never uses the same bank as the procedure it interrupts. The USING attribute of a procedure, or the \$REGISTERBANK control, can be used to ensure that.

The SSD module uses the \$REGISTERBANK(1) attribute. Some procedures are modified with the USING attribute based on the register bank level of the calling procedure.

### 19.2.5 APPLICATION MODULE; Async to SDLC protocol converter

One of the purposes of this application module is to demonstrate how to interface software to the SSD. Another purpose is to implement and test a practical application. This application software performs I/O with an async terminal through a USART, buffers data, and also performs I/O with the SSD. In addition, it allows the user on the async terminal to: set the station ad-

dress, set the destination address, and go online and offline. Setting the station address sets the byte in the STAD register. The destination address is the first byte in the I field. Going online or offline results in either calling the OPEN or CLOSE procedure respectively. After the secondary station powers up, it enters the 'terminal mode', which accepts data from the terminal. However, before any data is sent, the user must configure the station. The station address and destination address must be set, and the station must be placed online. To configure the station the ESC character is entered at the terminal which puts the protocol converter into the 'configure mode'. Figure 20-13 shows the menu which appears on the terminal screen.

### ( ) 8044 Secondary Station

- 1 - Set the Station Address
- 2 - Set the Destination Address
- 3 - Go Online
- 4 - Go Offline
- 5 - Return to terminal mode

Enter option \_\_\_\_

**Figure 19-13. Menu for the Protocol Converter**

In the terminal mode data is buffered up in the secondary station. A Line Feed character 'LF' tells the secondary station to send an I frame. If more than 60 bytes are buffered in the secondary station when a 'LF' is received, the applications software packetizes the data into 60 bytes or less per frame. If a LF is entered when the station is offline, an error message comes on the screen which says 'Unable to Get Online'.

The secondary station also does error checking on the async interface for Parity, Framing Error, and Over-run Error. If one of these errors are detected, an error message is displayed on the terminal screen.

### Buffering

There are two separate buffers in the application module: a transmit buffer and a receive buffer. The transmit buffer receives data from the USART, and sends data to the SSD. The receive buffer receives data from the SSD, and transmits data to the USART. Each buffer is a 256 byte software FIFO. If the transmit FIFO becomes full and no 'LF' character is received, the secondary station automatically begins sending the data. In addition, the application module will shut off the terminal's transmitter using CTS until the FIFO has been partially emptied. A block diagram of the buffering for the protocol converter is given in Figure 19-14.

### Application Module Software

A block diagram of the application module software is given in Figure 19-15. There are three interrupt



routines in this module: USART\_RECV\_INT, USART\_XMIT\_INT, and TIMER\_0\_INT. The first two are for servicing the USART. TIMER\_0\_INT is used if the TRANSMIT procedure in the SSD is called and does not return with the DATA\_TRANSMITTED parameter. TIMER\_0\_INT employs Timer 0 to wait a finite amount of time before trying to transmit again. The highest priority interrupt is USART\_RECV\_INT. The main program and all the procedures it calls use register bank 0, USART\_XMIT\_INT and TIMER\_0\_INT and FIFO\_R\_OUT use bank 1, while USART\_RECV\_INT and all the procedures it calls use register bank 2.

### Power\_On Procedure

The Power\_On procedure initializes all of the chips in the system including the 8044. The 8044 is initialized to use the on-chip DPLL with NRZI coding, PreFrame Sync, and Timer 1 auto reload at a baud rate of 62.5 Kbps. The 8254 and the 8251A are initialized next based on the DIP switch values attached to port 1 on the 8044. Variables and pointers are initialized, then the SSD's Power-Up Procedure, Power\_On\_D, is called. Finally the interrupt system is enabled and the main program is entered.

### Main Program

The main program is a simple loop which waits for a frame transmit command. A frame transmit command is indicated when the variable SEND\_DATA is greater than 0. The value of SEND\_DATA equals the number of 'LF' characters in the transmit FIFO, hence it also indicates the number of frames pending transmission. Each time a frame is sent, SEND\_DATA is decremented by one. Thus when SEND\_DATA is greater than 0, the main program falls down into the next loop which polls the XMIT\_BUFFER\_EMPTY bit. When XMIT\_BUFFER\_EMPTY equals 1, the SIU\_XMIT\_BUFFER can be loaded. The first byte in the buffer is loaded with the destination address while the rest of the buffer is loaded with the data. Bytes are removed from the transmit FIFO and placed into the SIU\_XMIT\_BUFFER until one of three things happen: 1. a 'LF' character is read out of the FIFO, 2. the number of bytes loaded equals the size of the SIU\_XMIT\_BUFFER, or 3. the transmit FIFO is empty.

After the SIU\_XMIT\_BUFFER is filled, the SSD TRANSMIT procedure is called and the results from the procedure are checked. Any result other than DATA\_TRANSMITTED will result in several retries within a finite amount of time. If all the retries fail then the LINK\_DISC procedure is called which sends a message to the terminal, 'Unable to Get Online'.

### USART\_RECV\_INT Procedure

When the 8251A receives a character, the RxRDY pin

on the 8251A is activated, and this interrupt procedure is entered. The routine reads the USART status register to determine if there are any errors in the character received. If there are, the character is discarded and the ERROR procedure is called which prints the type of error on the screen. If there are no errors, the received character is checked to see if it's an ESC. If it is an ESC, the MENU procedure is called which allows the user to change the configuration. If neither one of these two conditions exists the received character is inserted into the transmit FIFO. The received character may or may not be echoed back to the terminal based on the dip switch settings.

### Transmit FIFO

The transmit FIFO consists of two procedures: FIFO\_T\_IN and FIFO\_T\_OUT. FIFO\_T\_IN inserts a character into the FIFO, and FIFO\_T\_OUT removes a character from the FIFO. The FIFO itself is an array of 256 bytes called FIFO\_T. There are two pointers used as indexes in the array to address the characters: IN\_PTR\_T and OUT\_PTR\_T. IN\_PTR\_T points to the location in the array which will store the next byte of data inserted. OUT\_PTR\_T points to the next byte of data removed from the array. Both IN\_PTR\_T and OUT\_PTR\_T are declared as bytes. The FIFO\_T\_IN procedure receives a character from the USART\_RECV\_INT procedure and stores it in the array location pointed to by IN\_PTR\_T, then IN\_PTR\_T is incremented. Similarly, when FIFO\_T\_OUT is called by the main program, to load the SIU\_XMIT\_BUFFER, the byte in the array pointed to by OUT\_PTR\_T is read, then OUT\_PTR\_T is incremented. Since IN\_PTR\_T and OUT\_PTR\_T are always incremented, they must be able to roll over when they hit the top of the 256 byte address space. This is done automatically by having both IN\_PTR\_T and OUT\_PTR\_T declared as bytes. Each character inserted into the transmit FIFO is tested to see if it's a LF. If it is a LF, the variable SEND\_DATA is incremented which lets the main program know that it is time to send an I frame. Similarly each character removed from the FIFO is tested. SEND\_DATA is decremented for every LF character removed from the FIFO.

IN\_PTR\_T and OUT\_PTR\_T are also used to indicate how many bytes are in the FIFO, and whether it is full or empty. When a character is placed into the FIFO and IN\_PTR\_T is incremented, the FIFO is full if IN\_PTR\_T equals OUT\_PTR\_T. When a character is read from the FIFO and OUT\_PTR\_T is incremented, the FIFO is empty if OUT\_PTR\_T equals IN\_PTR\_T. If the FIFO is neither full nor empty, then it is in use. A byte called BUFFER\_STATUS\_T is used to indicate one of these three conditions. The application module uses the buffer status information to control the flow of data into and out of the FIFO. When the transmit FIFO is empty, the main program must stop loading bytes into the SIU\_

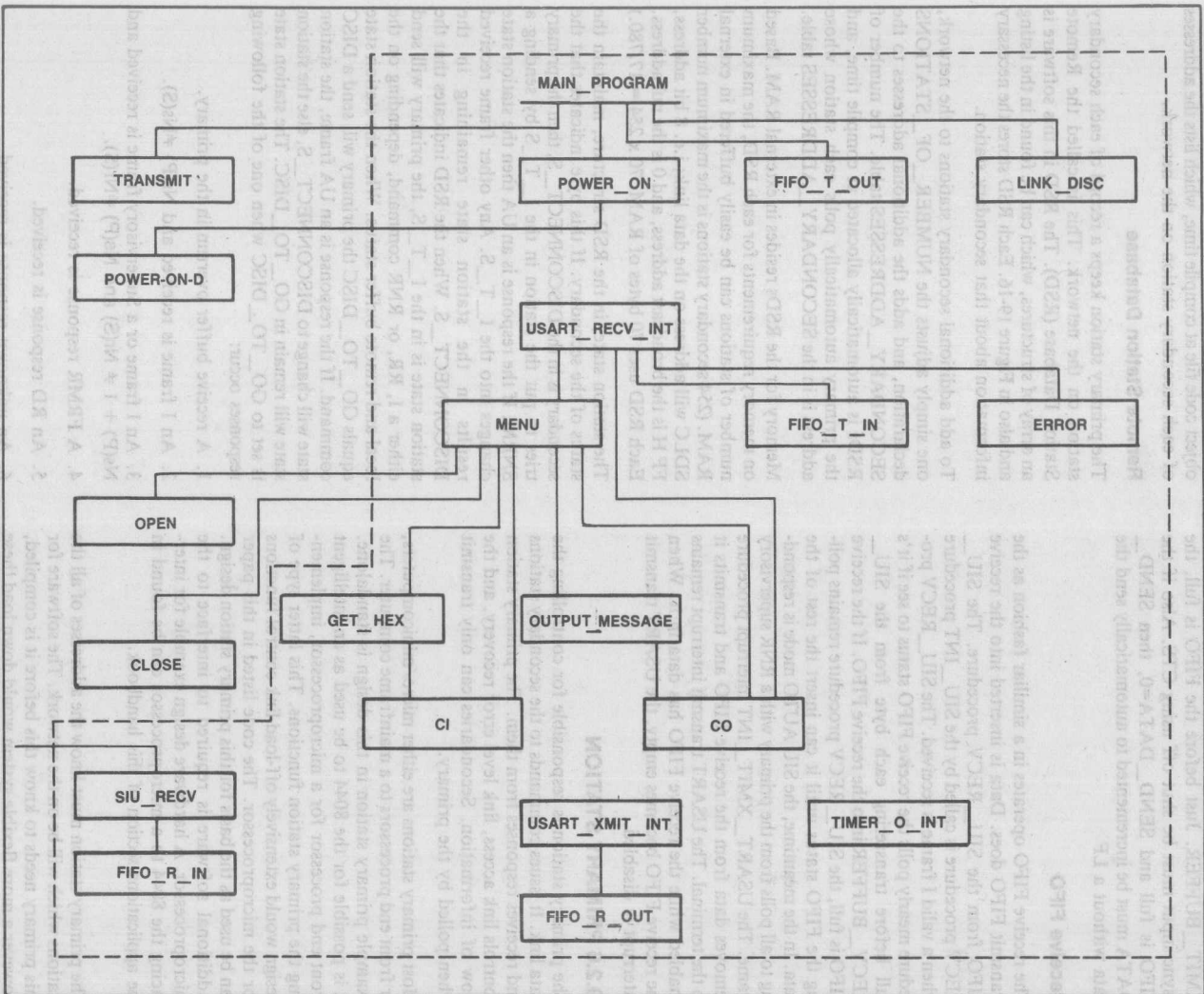


Figure 19-15. Block Diagram of User Software



XMIT\_BUFFER. Just before the FIFO is full, the async input must be shut off using CTS. Also if the FIFO is full and SEND\_DATA=0, then SEND\_DATA must be incremented to automatically send the data without a LF.

### Receive FIFO

The receive FIFO operates in a similar fashion as the transmit FIFO does. Data is inserted into the receive FIFO from the SIU\_RECV procedure. The SIU\_RECV procedure is called by the SIU\_INT procedure when a valid I frame is received. The SIU\_RECV procedure nearly polls the receive FIFO status to see if it's full before transferring each byte from the SIU\_RECV\_BUFFER into the receive FIFO. If the receive FIFO is full, the SIU\_RECV procedure remains polling the FIFO status until it can insert the rest of the data. In the meantime, the SIU AUTO mode is responding to all polls from the primary with a RNR supervisory frame. The USART\_XMIT\_INT interrupt procedure removes data from the receive FIFO and transmits it to the terminal. The USART transmit interrupt remains enabled while the receive FIFO has data in it. When the receive FIFO becomes empty, the USART transmit interrupt is disabled.

### 19.2.6 PRIMARY STATION

The primary station is responsible for controlling the data link. It issues commands to the secondary stations and receives responses from them. The primary station controls link access, link level error recovery, and the flow of information. Secondaries can only transmit when polled by the primary.

Most primary stations are either micro/minicomputers, or front end processors to a mainframe computer. The example primary station in this design is standalone. It is possible for the 8044 to be used as an intelligent front end processor for a microprocessor, implementing the primary station functions. This latter type of design would extensively off-load link control functions for the microprocessor. The code listed in this paper can be used as the basis for this primary station design. Additional software is required to interface to the microprocessor. A hardware design example for interfacing the 8044 to a microprocessor can be found in the applications section of this handbook.

The primary station must know the addresses of all the stations which will be on the network. The software for this primary needs to know this before it is compiled, however a more flexible system would download these parameters.

From the listing of the software it can be seen that the variable NUMBER\_OF\_STATIONS is a literal declaration, which is 2 in this design example. There were three stations tested on this data link, two secondaries and one primary. Following the NUMBER\_OF\_STATIONS declaration is a table, loaded into the

object code file at compile time, which lists the addresses of each secondary station on the network.

### Remote Station Database

The primary station keeps a record of each secondary station on the network. This is called the Remote Station Database (RSD). The RSD in this software is an array of structures, which can be found in the listing and also in Figure 19-16. Each RSD stores the necessary information about that secondary station.

To add additional secondary stations to the network, one simply adjusts the NUMBER\_OF\_STATIONS declaration, and adds the additional addresses to the SECONDARY\_ADDRESSES table. The number of RSDs is automatically allocated at compile time, and the primary automatically polls each station whose address is in the SECONDARY\_ADDRESSES table.

Memory for the RSDs resides in external RAM. Based on memory requirements for each RSD, the maximum number of stations can be easily buffered in external RAM. (254 secondary stations is the maximum number SDLC will address on the data link; i.e. 8 bit address, FF H is the broadcast address, and 0 is the nul address. Each RSD uses 70 bytes of RAM.  $70 \times 254 = 17,780$ .)

The station state, in the RSD structure, maintain the status of the secondary. If this byte indicates that the secondary is in the DISCONNECT\_S, then the primary tries to put the station in the I\_T\_S by sending a SNRM. If the response is an UA then the station state changes into the I\_T\_S. Any other frame received results in the station state remaining in the DISCONNECT\_S. When the RSD indicates that the station state is in the I\_T\_S, the primary will send either a I, RR, or RNR command, depending on the local and remote buffer status. When the station state equals GO\_TO\_DISC the primary will send a DISC command. If the response is an UA frame, the station state will change to DISCONNECT\_S, else the station state will remain in GO\_TO\_DISC. The station state is set to GO\_TO\_DISC when one of the following responses occur:

1. A receive buffer overrun in the primary.
2. An I frame is received and  $Nr(P) \neq Ns(S)$ .
3. An I frame or a Supervisory frame is received and  $Ns(P) + 1 \neq Nr(S)$  and  $Ns(P) \neq Nr(S)$ .
4. A FRMR response is received.
5. An RD response is received.
6. An unknown response is received.

The send count (Ns) and receive count (Nr) are also maintained in the RSD. Each time an I frame is sent by the primary and acknowledged by the secondary, Ns is incremented. Nr is incremented each time a valid I frame is received. BUFFER\_STATUS indicates the status of the secondary stations buffer. If a RR response is received, BUFFER\_STATUS is set to BUFFER\_

READY. If a RNR response is received, BUFFER\_STATUS is set to BUFFER\_NOT\_READY.

### Buffering

The buffering for the primary station is as follows: within each RSD is a 64 byte array buffer which is initially empty. When the primary receives an I frame, it looks for a match between the first byte of the I frame and the addresses of the secondaries on the network. If a match exists, the primary places the data in the RSD buffer of the destination station. The INFO\_LENGTH in the RSD indicates how many bytes are in the buffer. If INFO\_LENGTH equals 0, then the buffer is empty. The primary can buffer only one I frame per station. If a second I frame is received while the addressed secondary's RSD buffer is full, the primary cannot receive any more I frames. At this point the primary continues to poll the secondaries using RNR supervisory frame.

### Primary Station Software

A block diagram of the primary station software is shown in Figure 19-17. The primary station software consists of a main program, one interrupt routine, and

several procedures. The POWER\_ON procedure begins by initializing the SIU's DMA and enabling the receiver. Then each RSD is initialized. The DPLL and the timers are set, and finally the TIMER 0 interrupt is enabled.

The main program consists of an iterative do loop within a do forever loop. The iterative do loop polls each secondary station once through the do loop. The variable STATION\_NUMBER is the counter for the iterative do statement which is also used as an index to the array of RSD structures. The primary station issues one command and receives one response from every secondary station each time through the loop. The first statement in the loop loads the secondary station address, indexed by STATION\_NUMBER into the array of the RSD structures. Now when the primary sends a command, it will have the secondary's address in the address field of the frame. The automatic address recognition feature is used by the primary to recognize the response from the secondary.

Next the main program determines the secondary stations state. Based on this state, the primary knows what command to send. If the station is in the DISCONNECT\_S, the primary calls the SNRM\_P

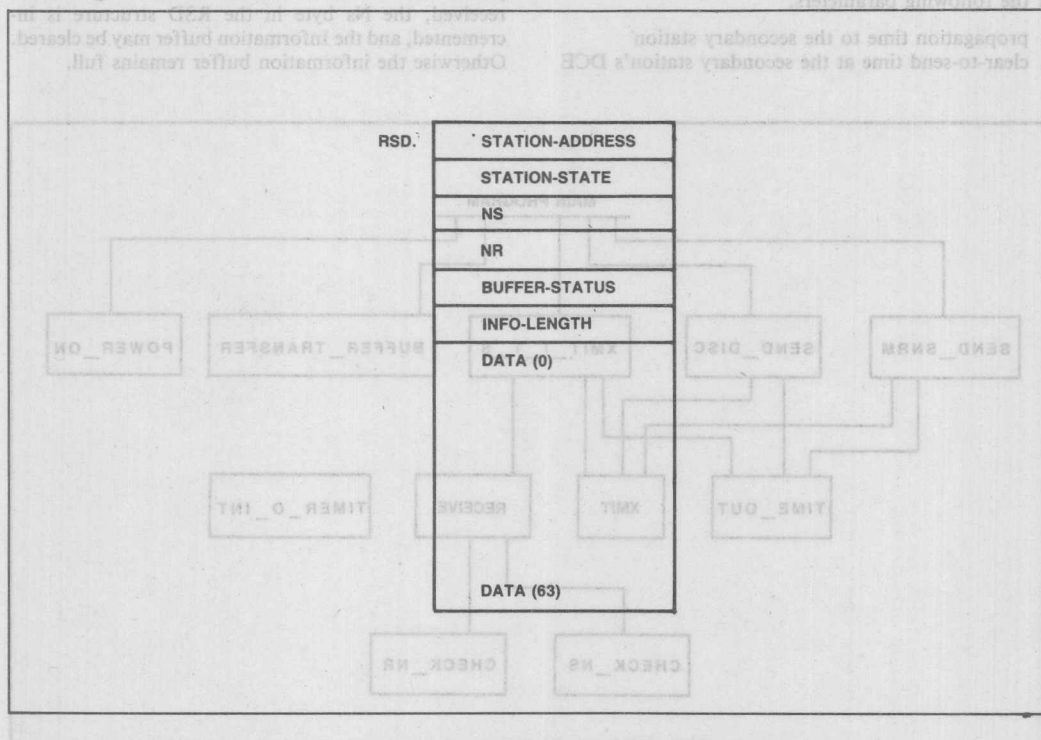


Figure 19-16. Remote Station Database Structure

procedure to try and put the secondary in the I\_T\_S. If the station state is in the GO\_TO\_DISC state, the DISC\_P is called to try and put the secondary in the L\_D\_S. If the secondary is in neither one of the above two states, then it is in the I\_T\_S. When the secondary is in the I\_T\_S, the primary could send one of three commands: I, RR, or RNR. If the RSD's buffer has data in it, indicated by INFO\_LENGTH being greater than zero, and the secondary's BUFFER\_STATUS equals BUFFER\_READY, then an I frame will be sent. Else if RPB=0, a RR supervisory frame will be sent. If neither one of these cases is true, then an RNR will be sent. The last statement in the main program checks the RPB bit. If set to one, the BUFFER\_TRANSFER procedure is called, which transfers the data from the SIU receive buffer to the appropriate RSD buffer.

#### Receive Time Out

Each time a frame is transmitted, the primary sets a receive time out timer; Timer 0. If a response is not received within a certain time, the primary returns to the main program and continues polling the rest of the stations. The minimum length of time the primary should wait for a response can be calculated as the sum of the following parameters.

1. propagation time to the secondary station
2. clear-to-send time at the secondary station's DCE

3. appropriate time for secondary station processing
4. propagation time from the secondary station
5. maximum frame length time

The clear-to-send time and the propagation time are negligible for a local network at low bit rates. However, the turnaround time and the maximum frame length time are significant factors. Using the 8044 secondaries in the AUTO mode minimizes turnaround time. The maximum frame length time comes from the fact the 8044 does not generate an interrupt from a received frame until it has been completely received, and the CRC is verified as correct. This means that the time-out is bit rate dependent.

#### Ns and Nr check Procedures

Each time an I frame or supervisory frame is received, the Nr field in the control byte must be checked. Since this data link only allows one outstanding frame, a valid Nr would satisfy either one of two equations;  $Ns(P) + 1 = Nr(S)$  the I frame previously sent by the primary is acknowledged,  $Ns(P) = Nr(S)$  the I frame previously sent is not acknowledged. If either one of these two cases is true, the CHECK\_NR procedure returns a parameter of TRUE; otherwise a FALSE parameter is returned. If an acknowledgement is received, the Ns byte in the RSD structure is incremented, and the Information buffer may be cleared. Otherwise the information buffer remains full.

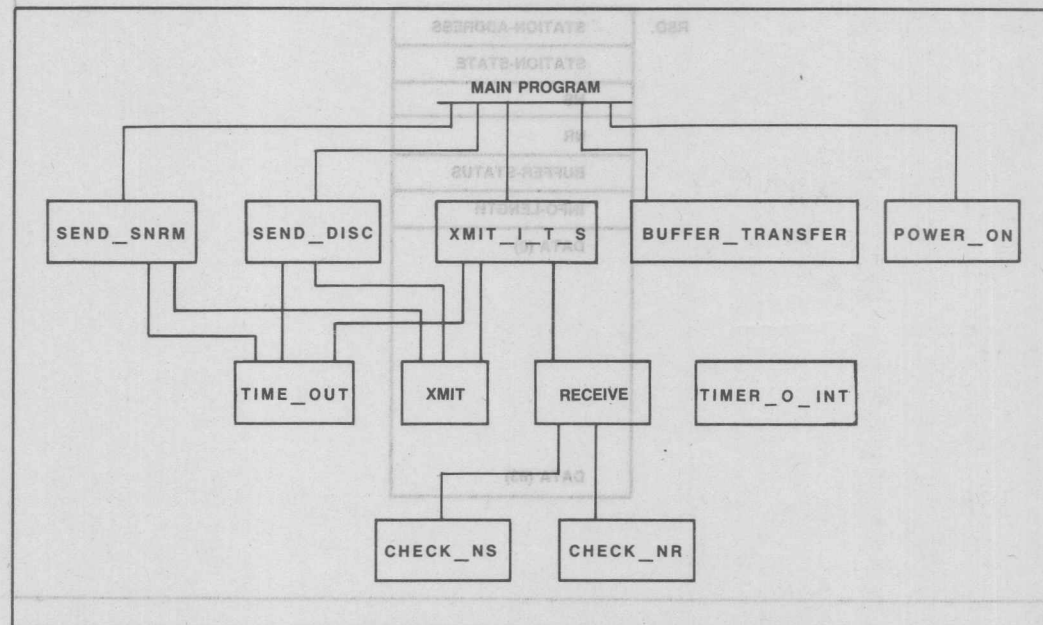


Figure 19-17. Block Diagram of Primary Station Software Structure

When an I frame is received, the Ns field has to be checked also. If  $Nr(P) = Ns(S)$ , then the procedure returns TRUE, otherwise a FALSE is returned.

### Receive Procedure

The receive procedure is called when a supervisory or information frame is sent, and a response is received before the time-out period. The RECEIVE procedure can be broken down into three parts. The first part is entered if an I frame is received. When an I frame is received, Ns, Nr and buffer overrun are checked. If there is a buffer overrun, or there is an error in either Ns or Nr, then the station state is set to GO\_TO\_DISC. Otherwise Nr in the RSD is incremented, the receive field length is saved, and the RPB bit is set. By incrementing the Nr field, the I frame just received is acknowledged the next time the primary polls the secondary with an I frame or a supervisory frame. Setting RBP protects the received data, and also tells

the main program that there is data to transfer to one of the RSD buffers.

If a supervisory frame is received, the Nr field is checked. If a FALSE is returned, then the station state is set to GO\_TO\_DISC. If the supervisory frame received was an RNR, buffer status is set to not ready. If the response is not an I frame, nor a supervisory frame, then it must be an Unnumbered frame.

The only Unnumbered frames the primary recognizes are UA, DM, and FRMR. In any event, the station state is set to GO\_TO\_DISC. However if the frame received is a FRMR, Nr in the second data byte of the I field is checked to see if the secondary acknowledged an I frame received before it went into the FRMR state. If this is not done and the secondary acknowledged an I frame which the primary did not recognize, the primary transmits, the I frame when the secondary returns to the I\_T\_S. In this case, the secondary would receive duplicate I frames.



the main program that there is data to transfer to one of the RSD buffers.

If a supervisory frame is received, the Nr field is checked. If a FALSE is returned, then the station state is set to GO\_TO\_DISC. If the supervisory frame received was an RNR, buffer status is set to not ready. If the response is not an I frame, not a supervisory frame, then it must be an Unnumbered frame.

The only Unnumbered frames the primary recognizes are UA, DM, and FRMR. In any event, the station state is set to GO\_TO\_DISC. However, if the frame received is a FRMR, Nr in the second data byte of the I field is checked to see if the secondary acknowledged an I frame. If it is not done and the secondary acknowledged an I frame, then the primary will not recognize the primary transmits the I frame when the secondary returns to the I\_T\_S. In this case, the secondary would receive duplicate I frames.

When an I frame is received, the Nr field has to be checked also. If Nr(P) = Nr(S), then the procedure returns TRUE, otherwise a FALSE is returned.

## Receive Procedure

The receive procedure is called when a supervisory or information frame is sent, and a response is received before the time-out period. The RECEIVE procedure can be broken down into three parts. The first part is entered if an I frame is received. When an I frame is received, Nr and buffer overrun are checked. If there is a buffer overrun, or there is an error in either Nr or Nr, then the station state is set to GO\_TO\_DISC. Otherwise Nr in the RSD is incremented by the receive field length is saved, and the RPB bit is set to 1. Incrementing the Nr field, then the primary polls the secondary acknowledged the next time the primary polls the secondary with an I frame or a supervisory frame. Setting RBP protects the received data, and also tells

# APPENDIX A 8044 SOFTWARE FLOWCHARTS

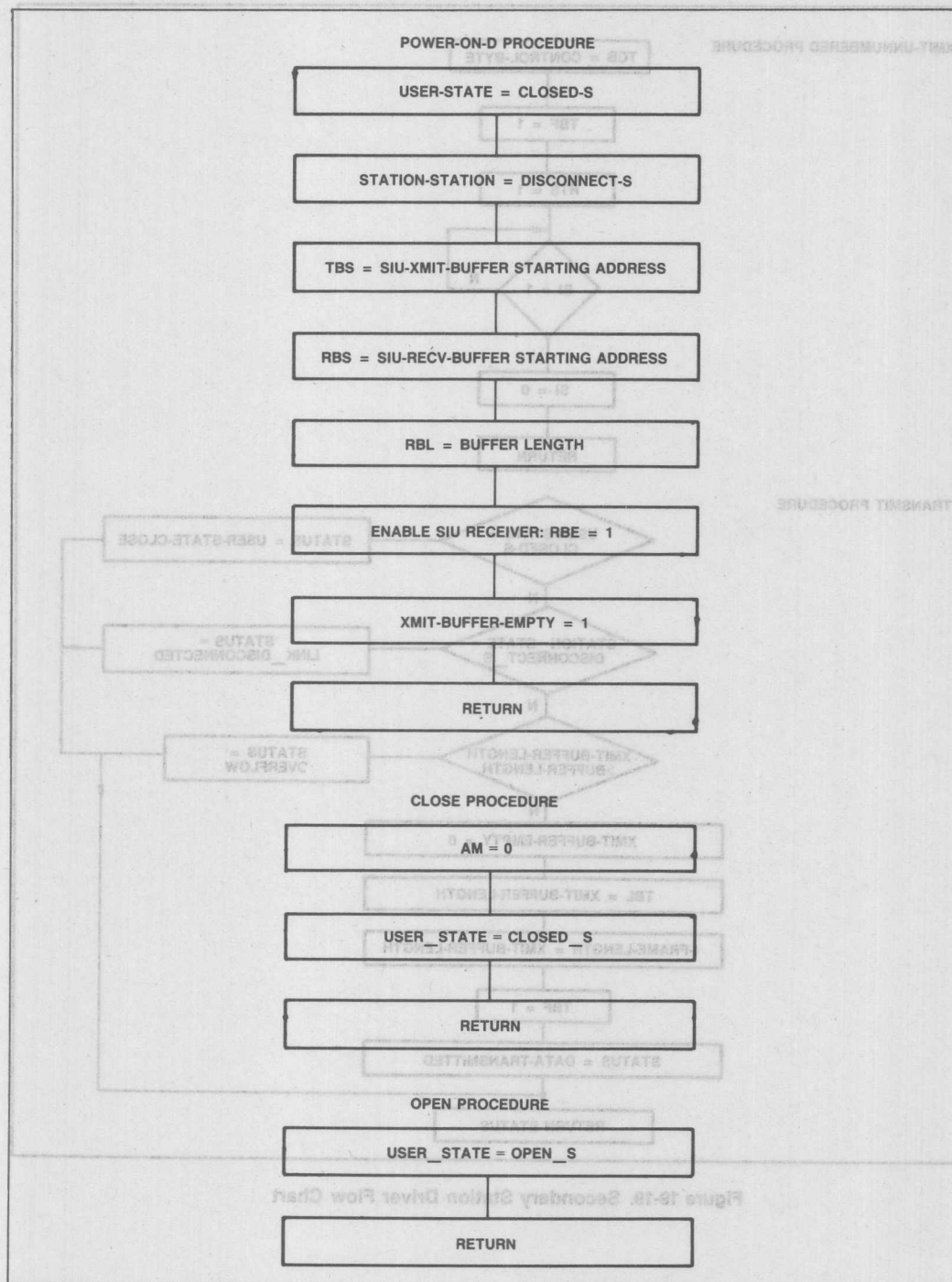


Figure 19-18. Secondary Station Driver Flow Chart

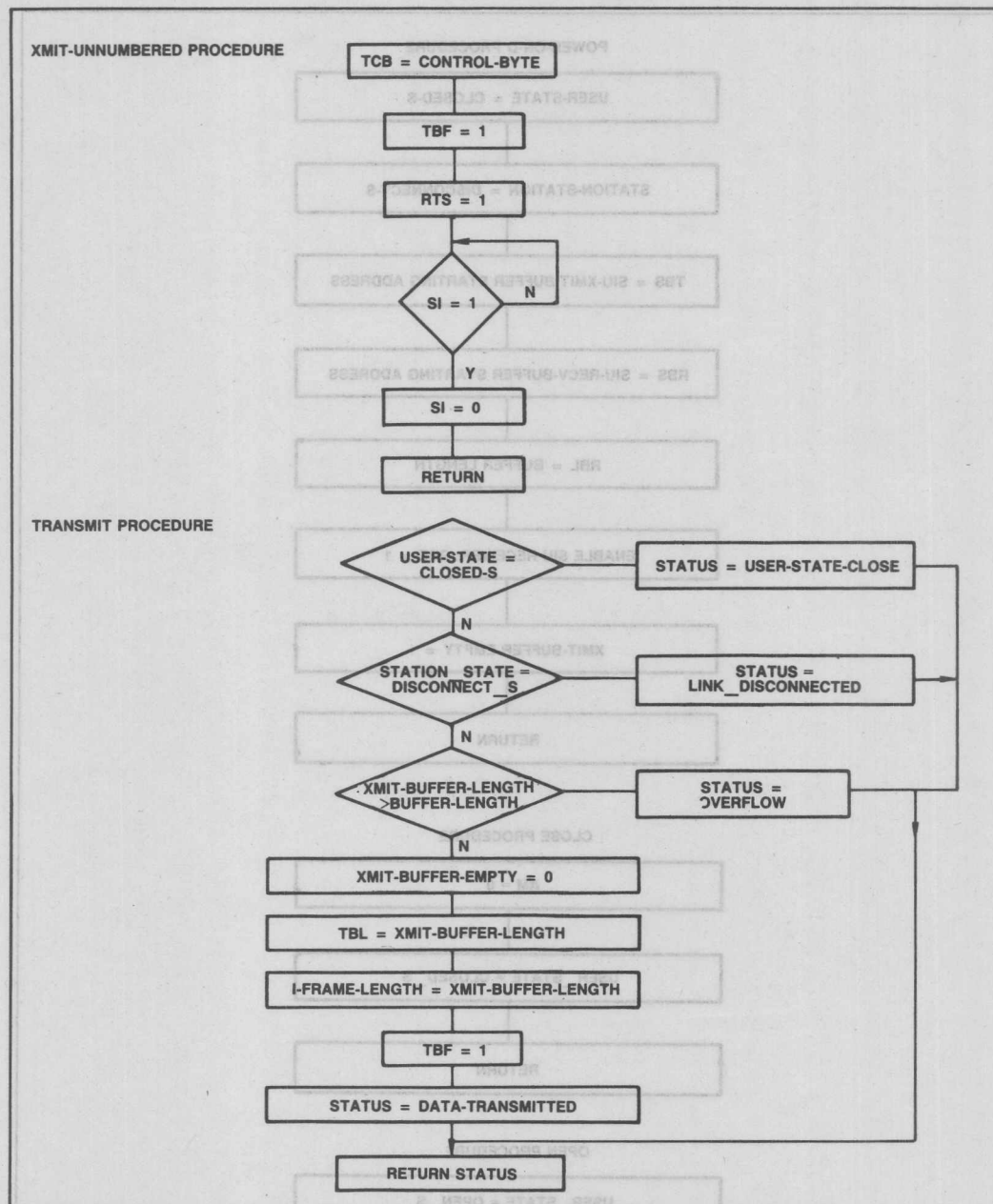


Figure 19-19. Secondary Station Driver Flow Chart

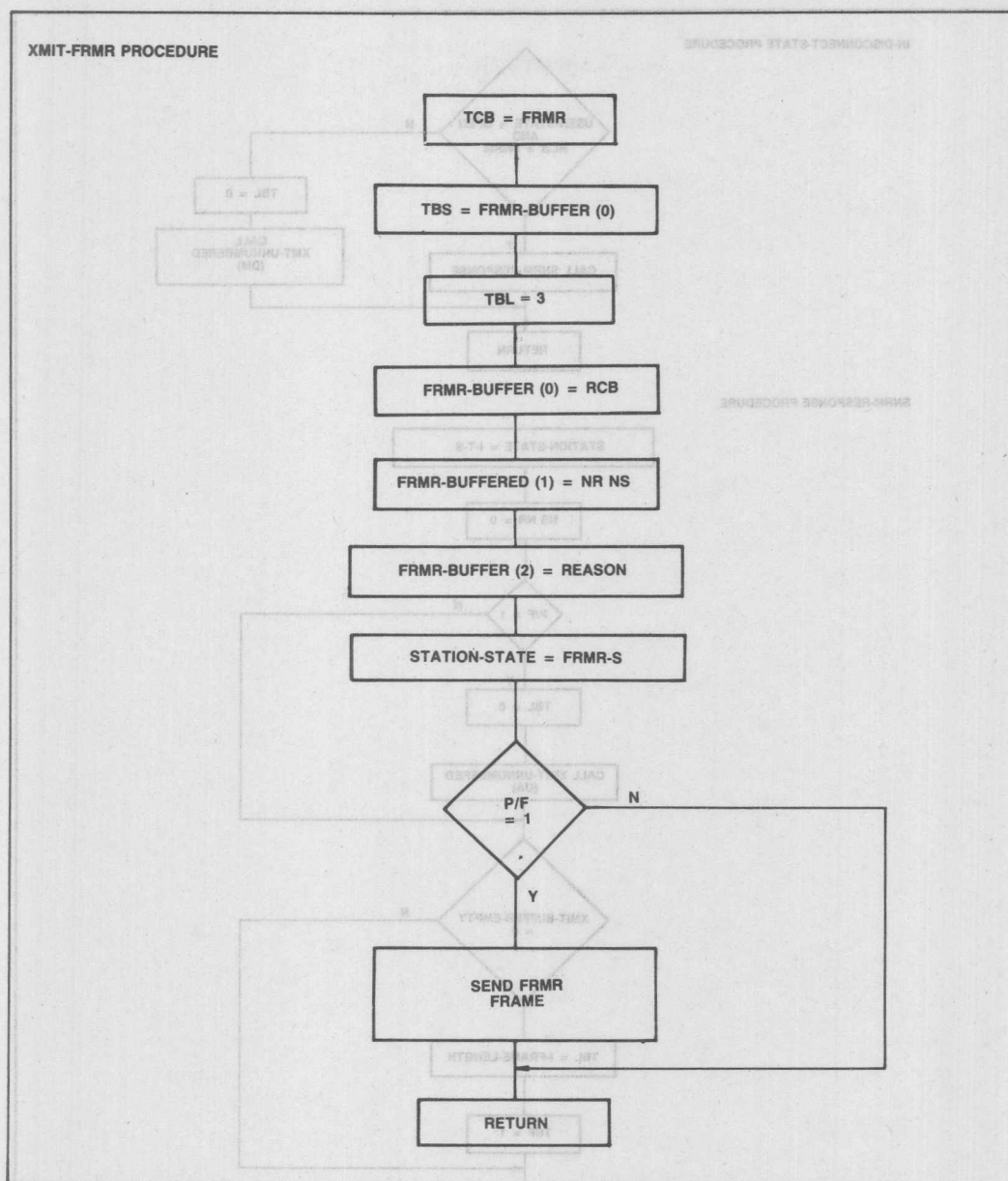


Figure 19-20. Secondary Station Driver Flow Chart



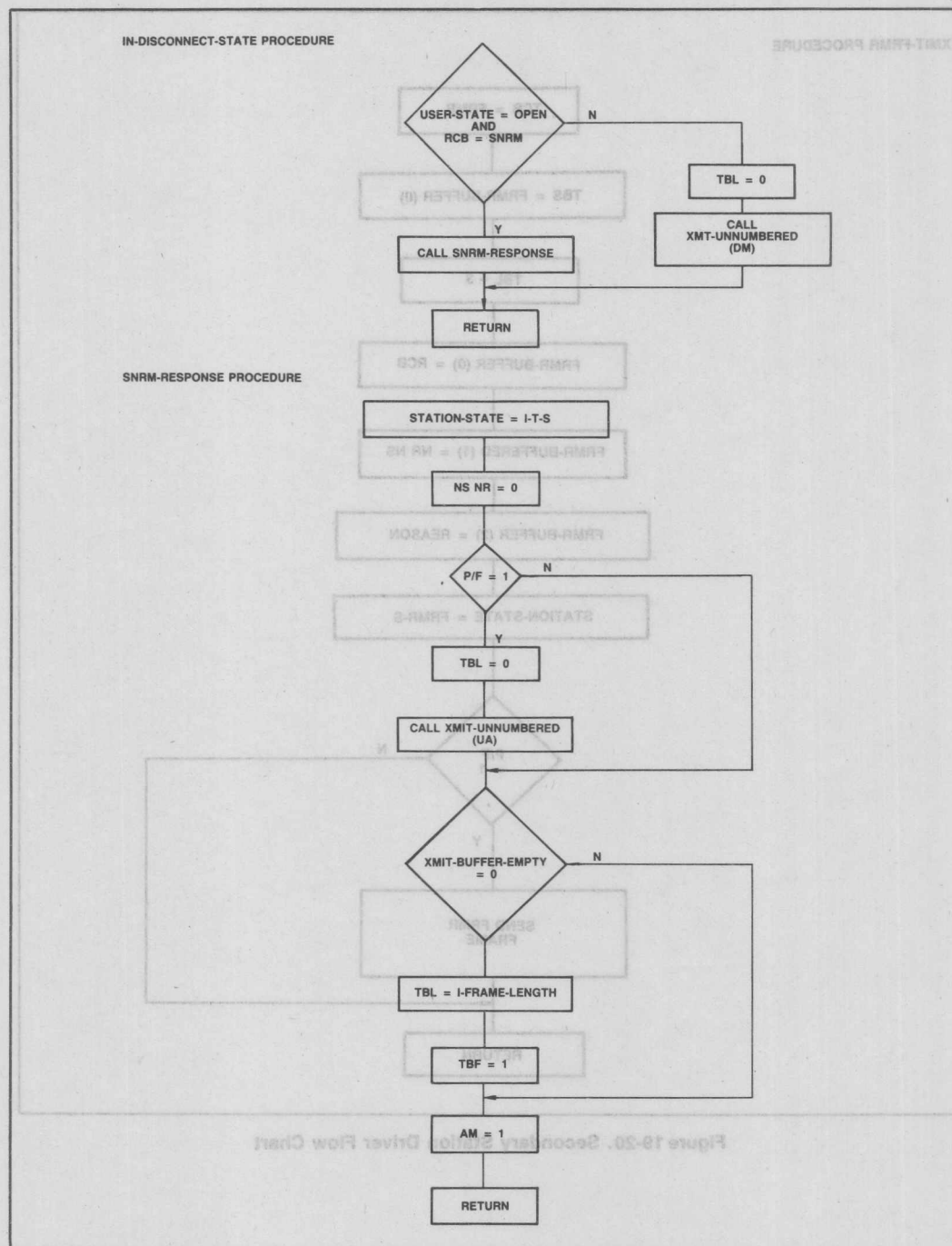


Figure 19-21. Secondary Station Driver Flow Chart

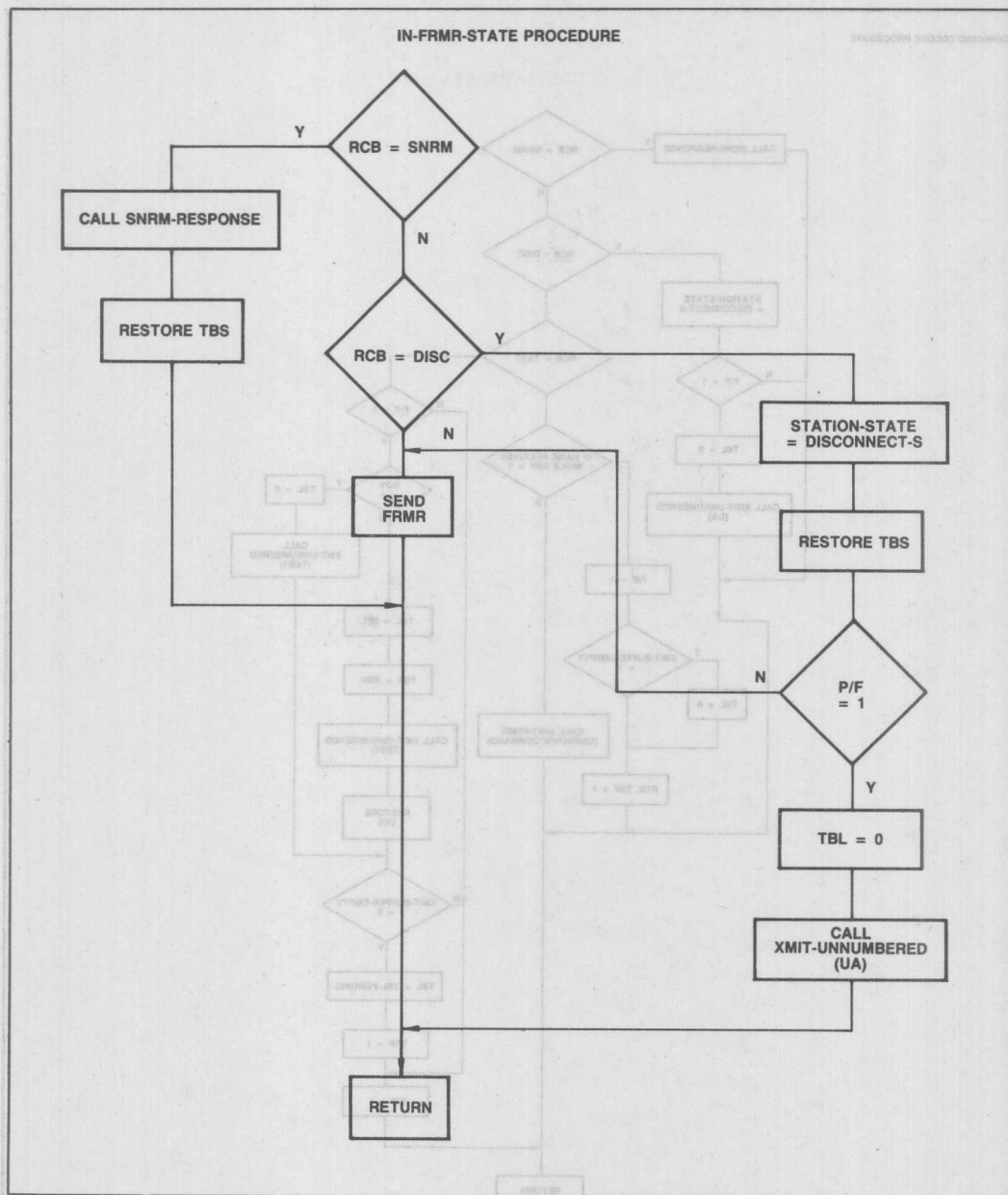


Figure 19-22 . Secondary Station Driver Flow Chart

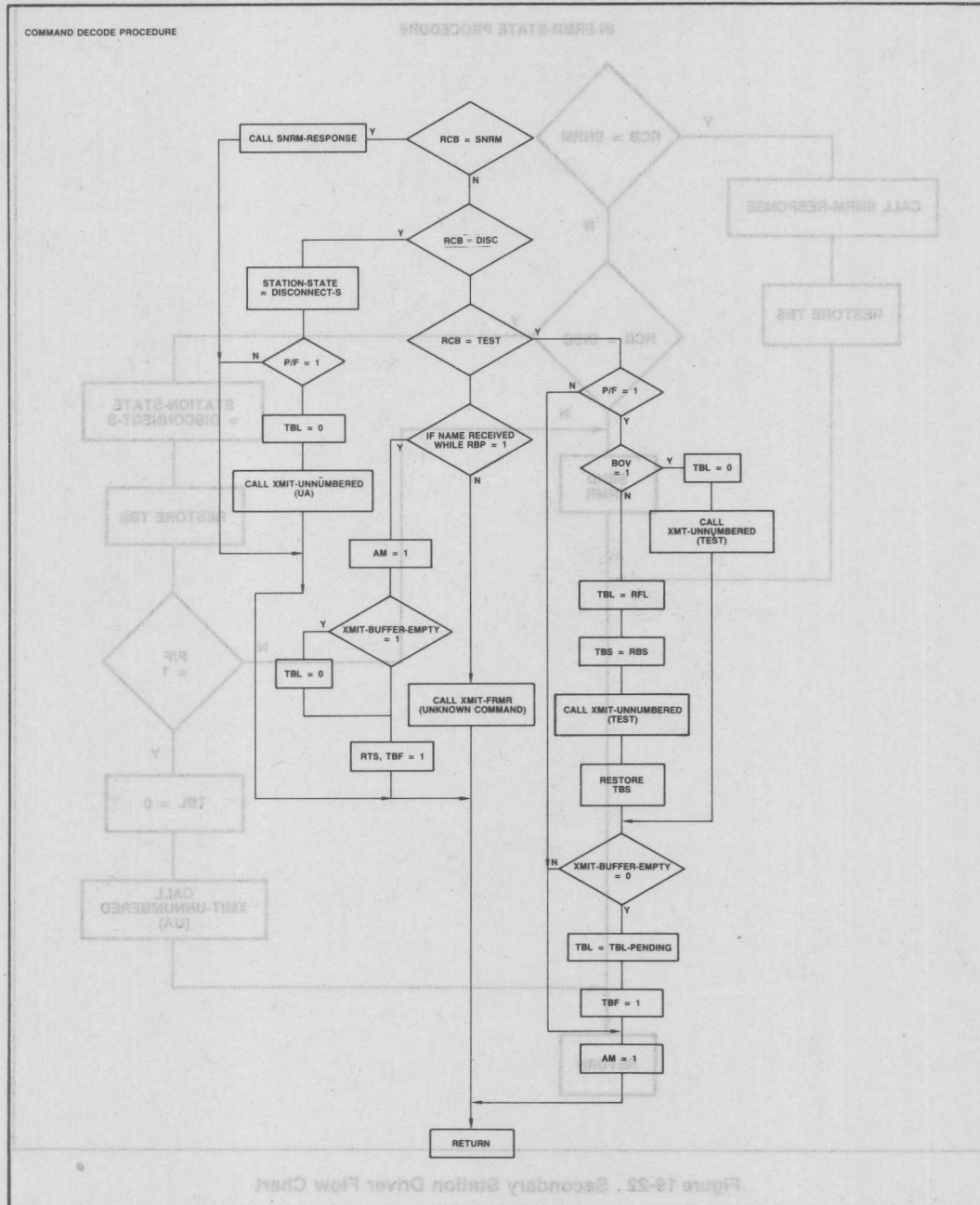


Figure 19-23. Secondary Station Driver Flow Chart

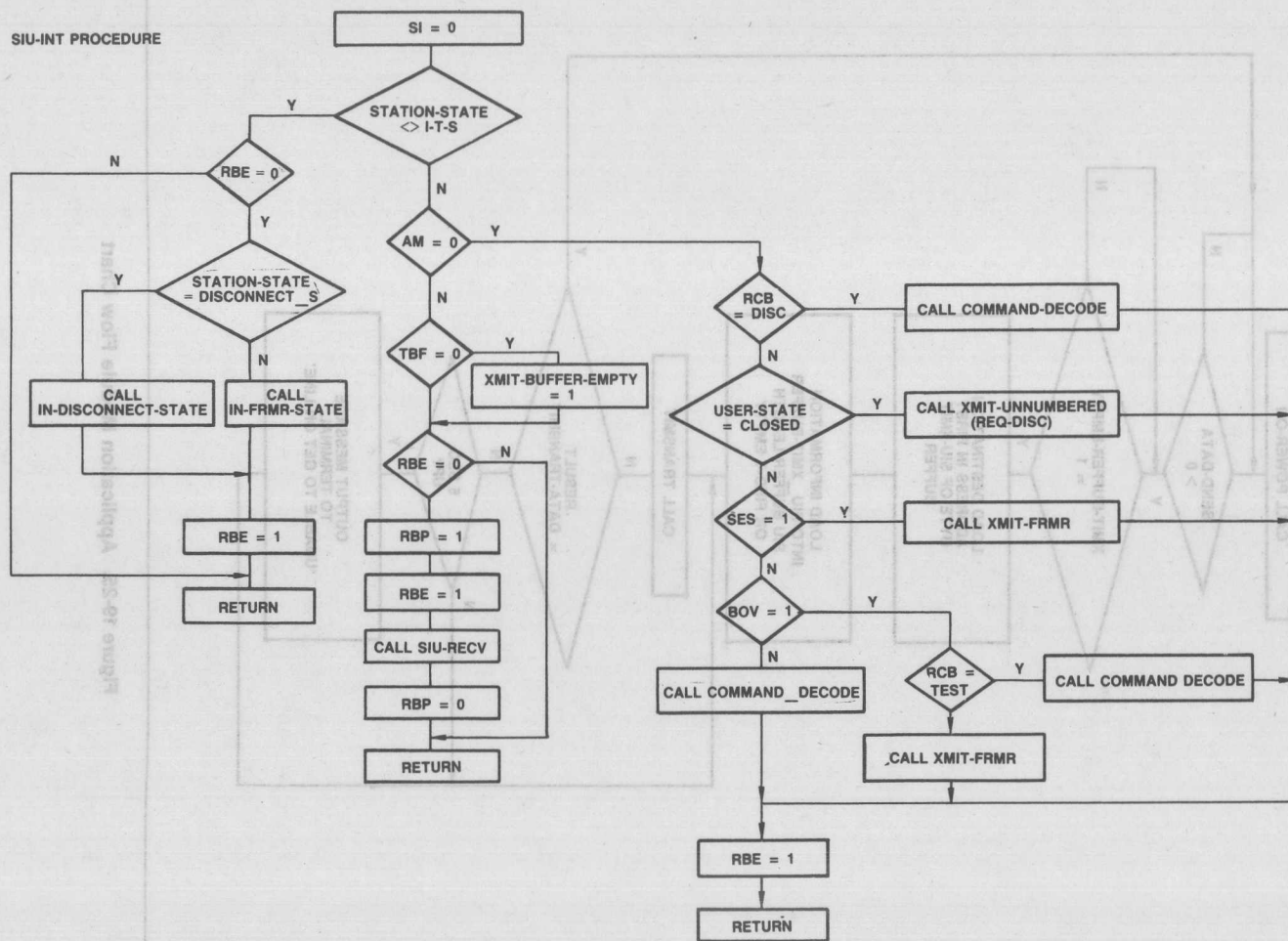


Figure 19-24. Secondary Station Driver Flow Chart



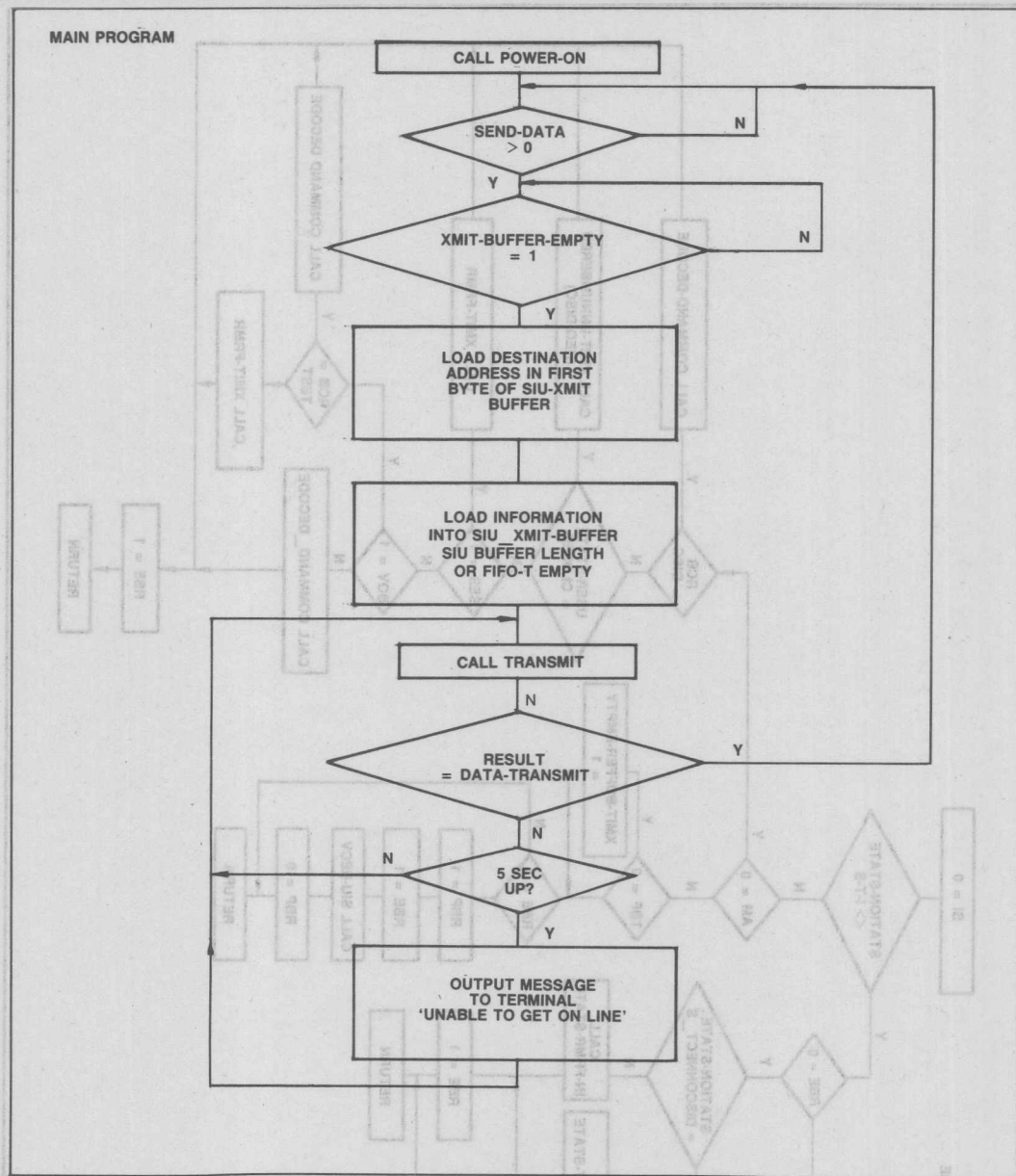


Figure 19-25. Application Module Flow Chart

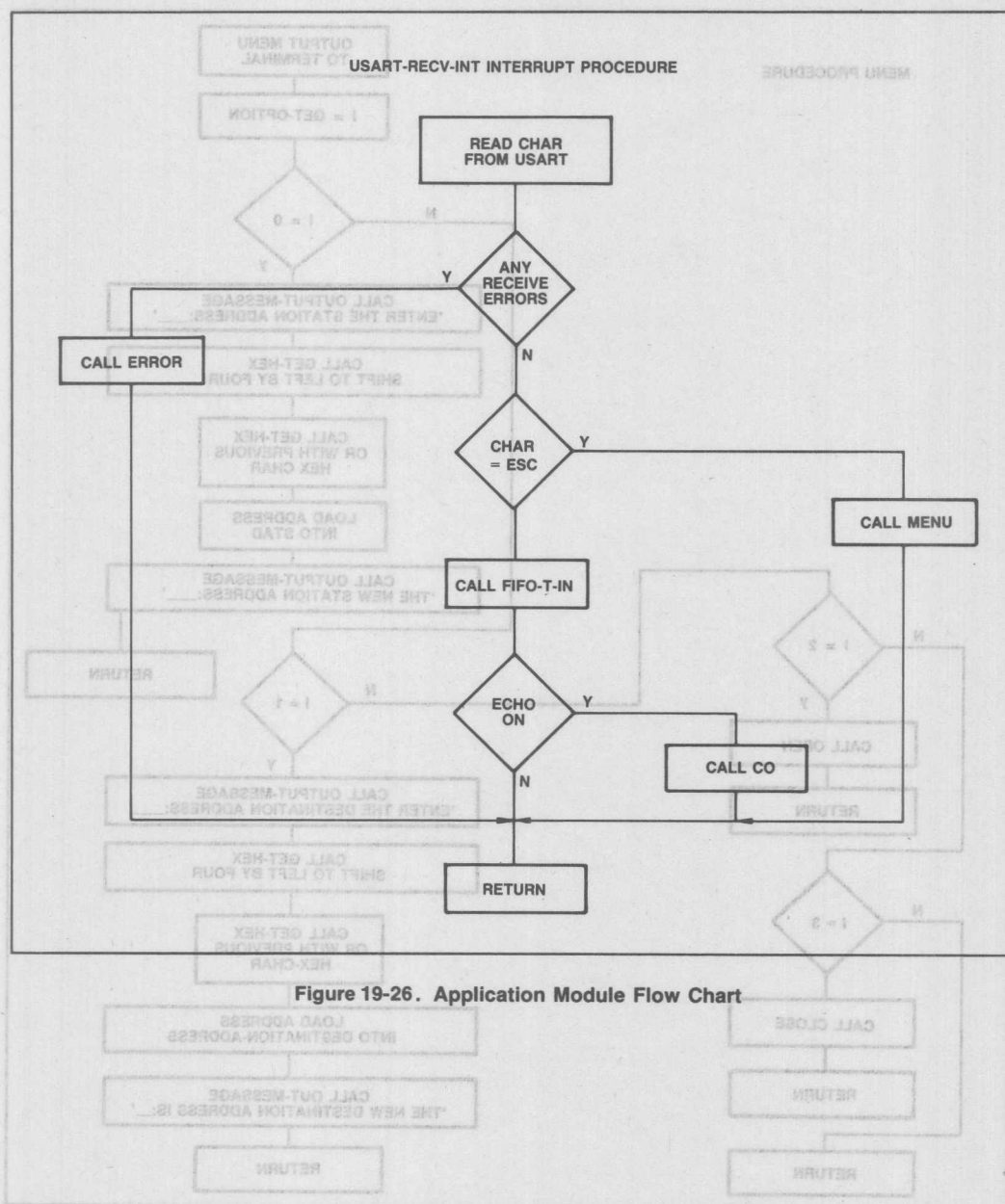


Figure 19-26. Application Module Flow Chart

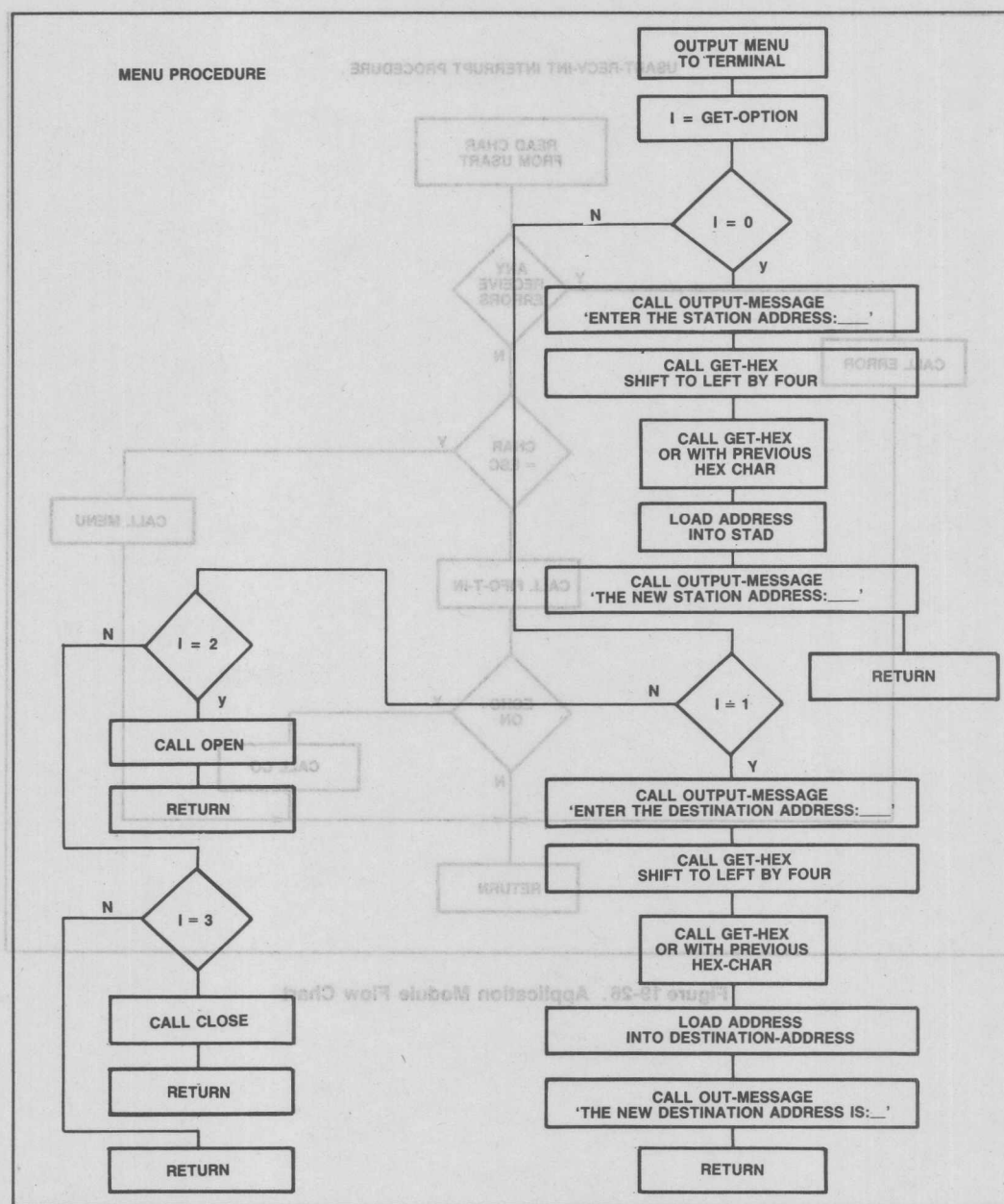


Figure 19-27. Application Module Flow Chart

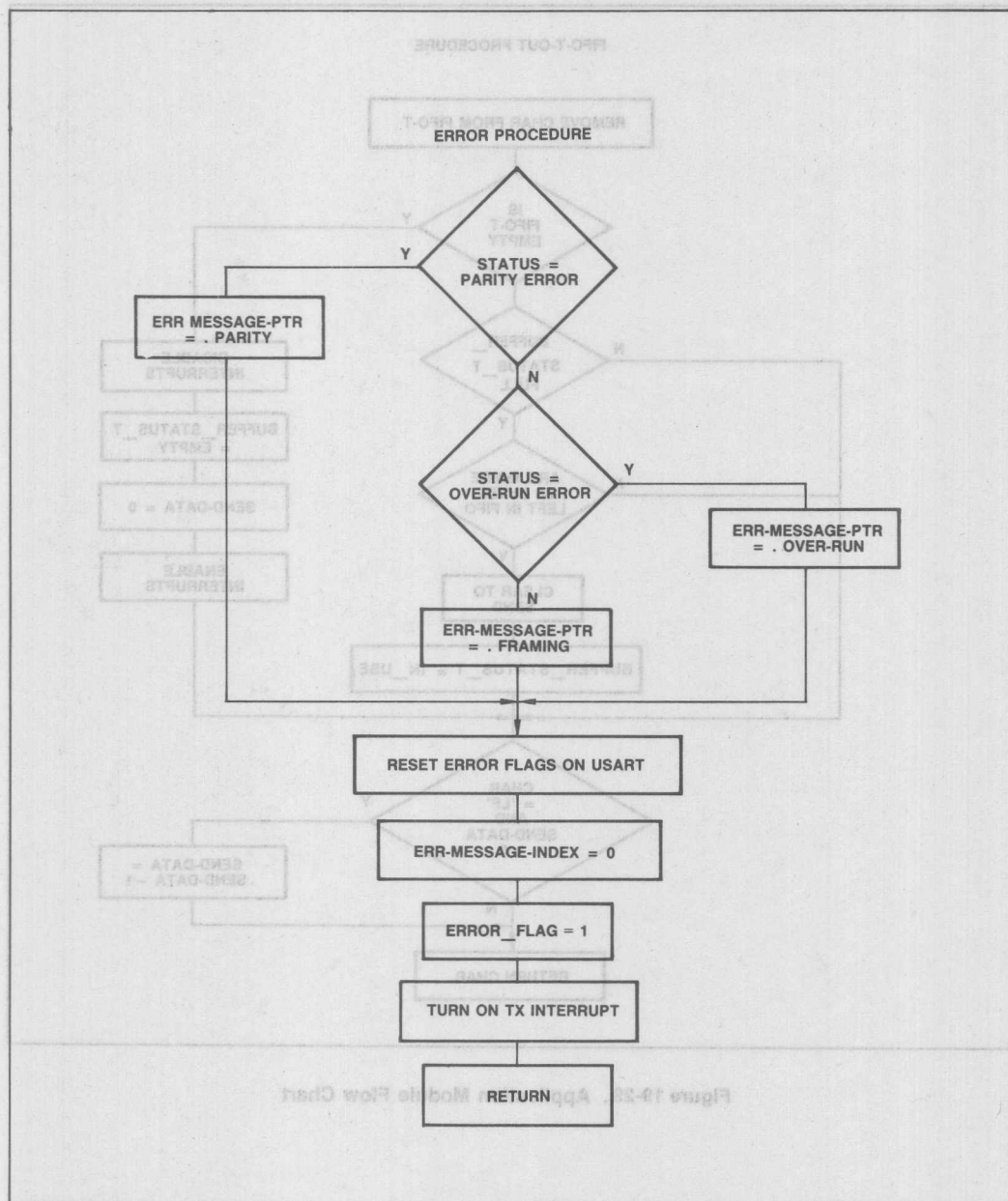


Figure 19-28 Application Module Flow Chart



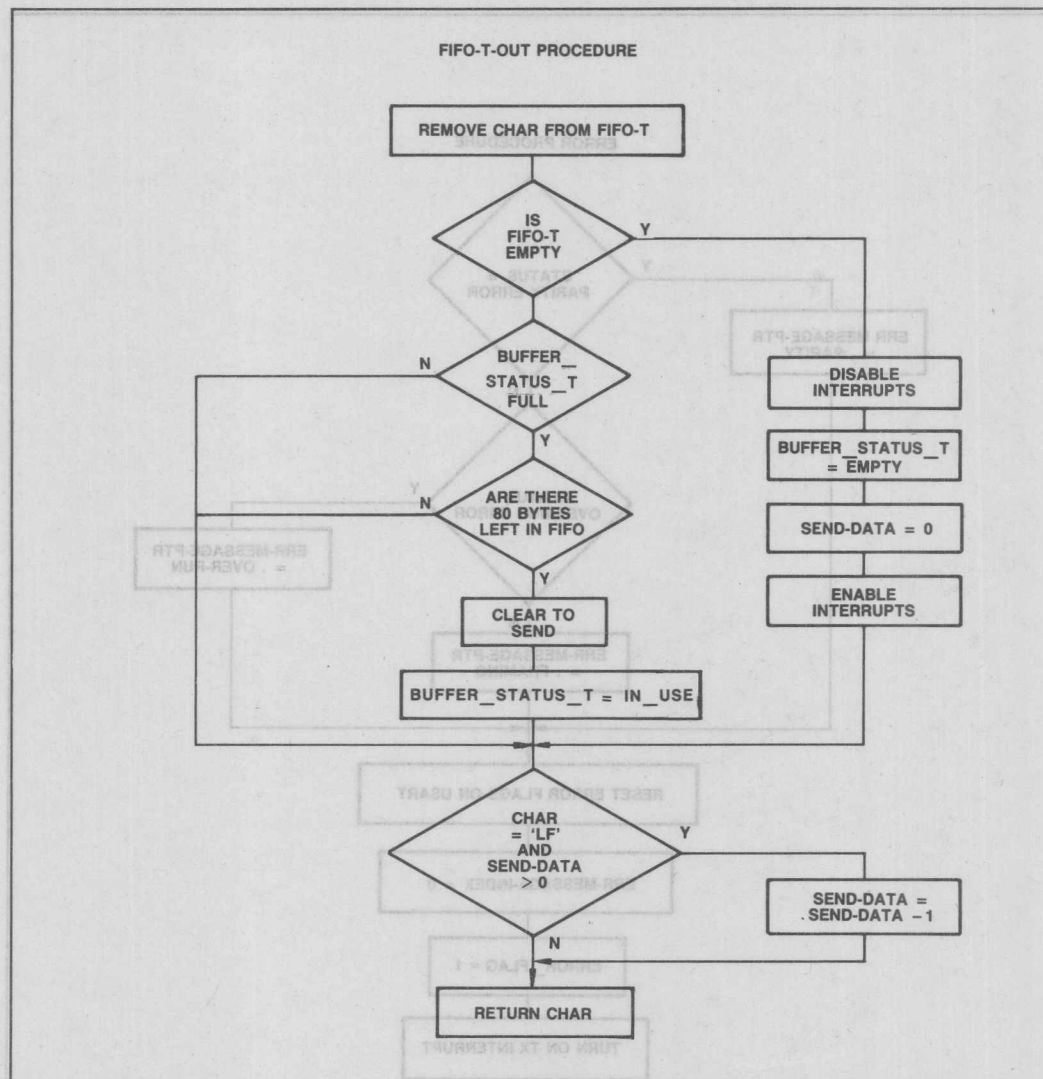


Figure 19-29. Application Module Flow Chart

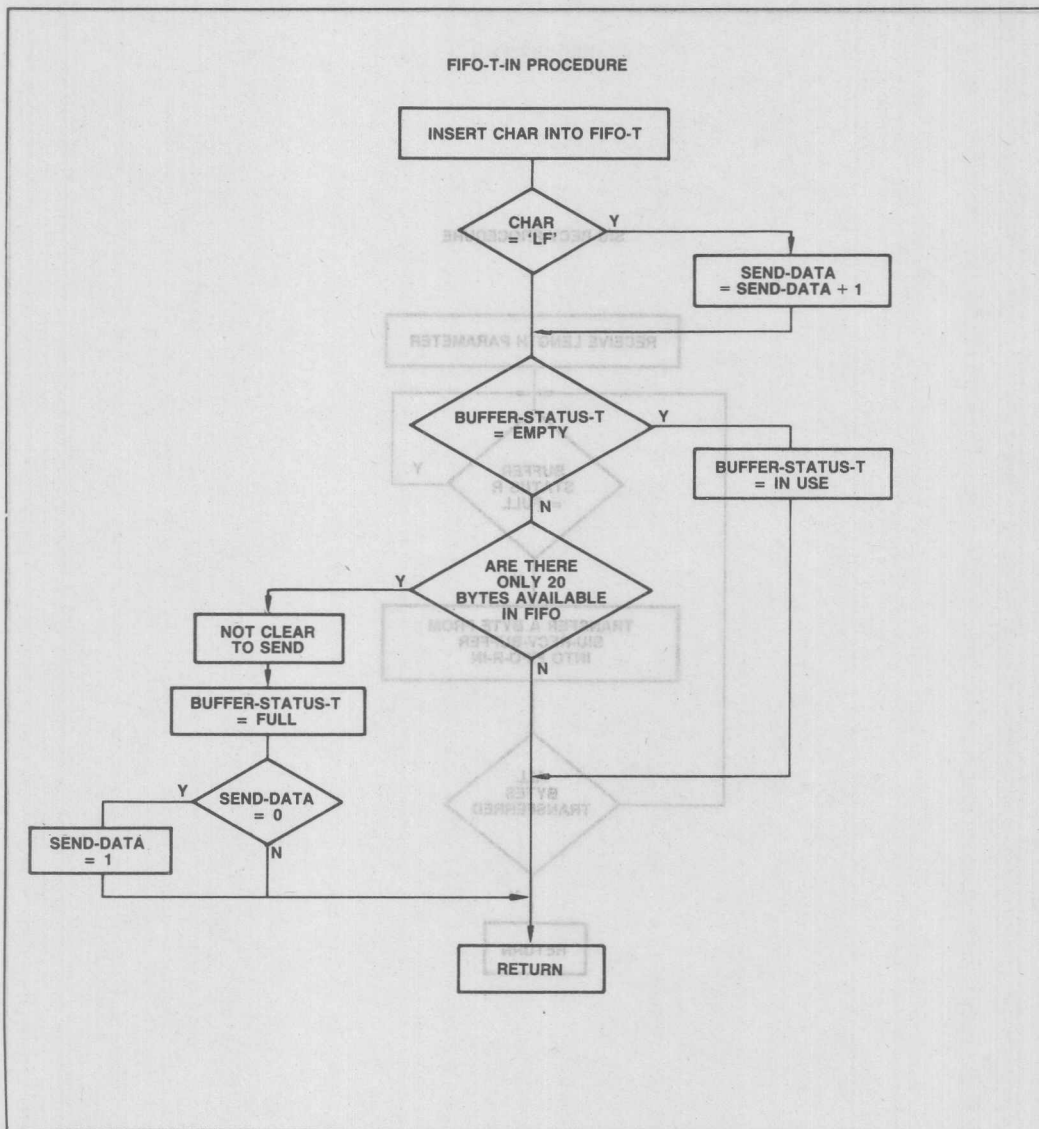


Figure 19-30. Application Module Flow Chart



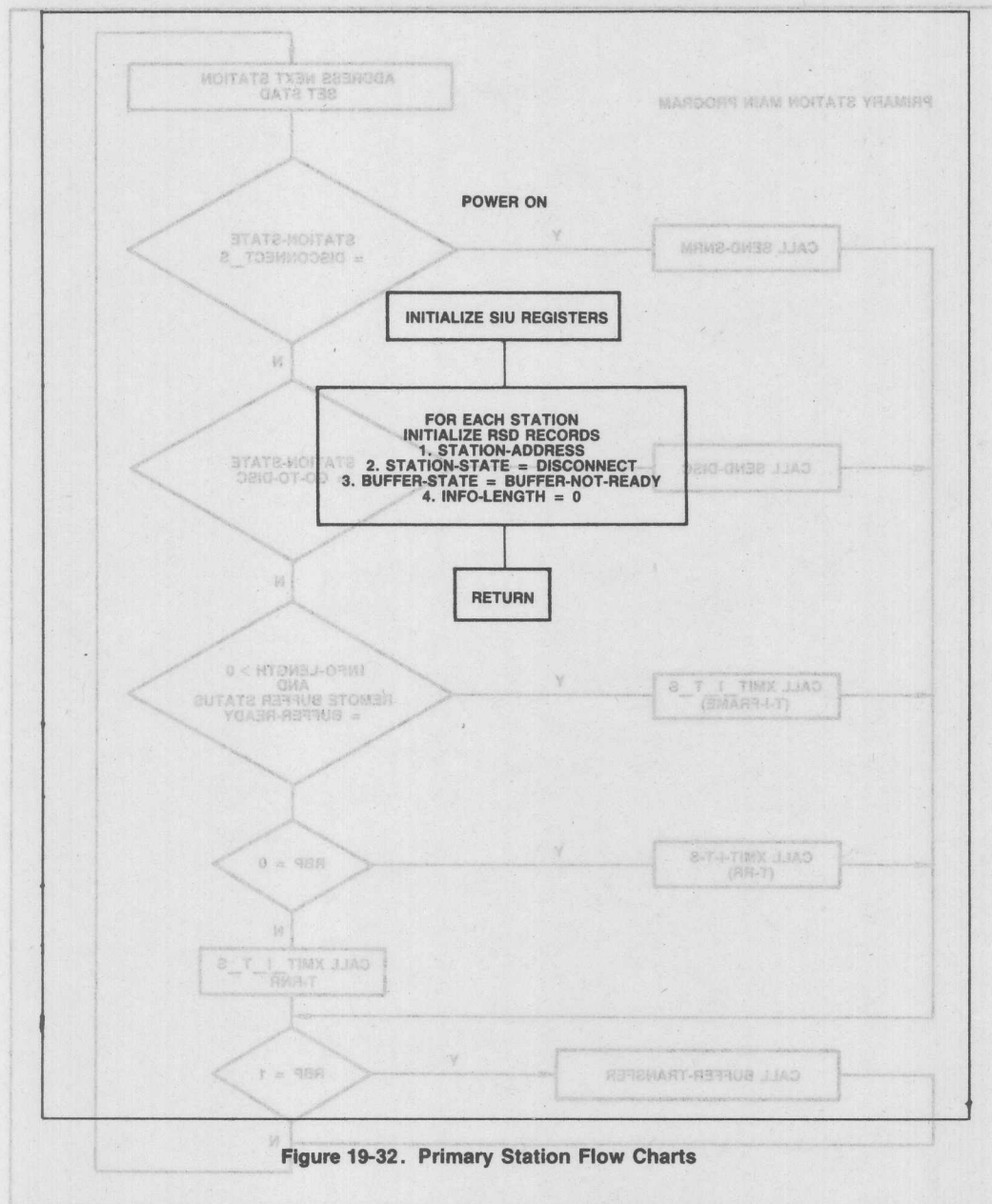


Figure 19-33. Primary Station Flow Charts

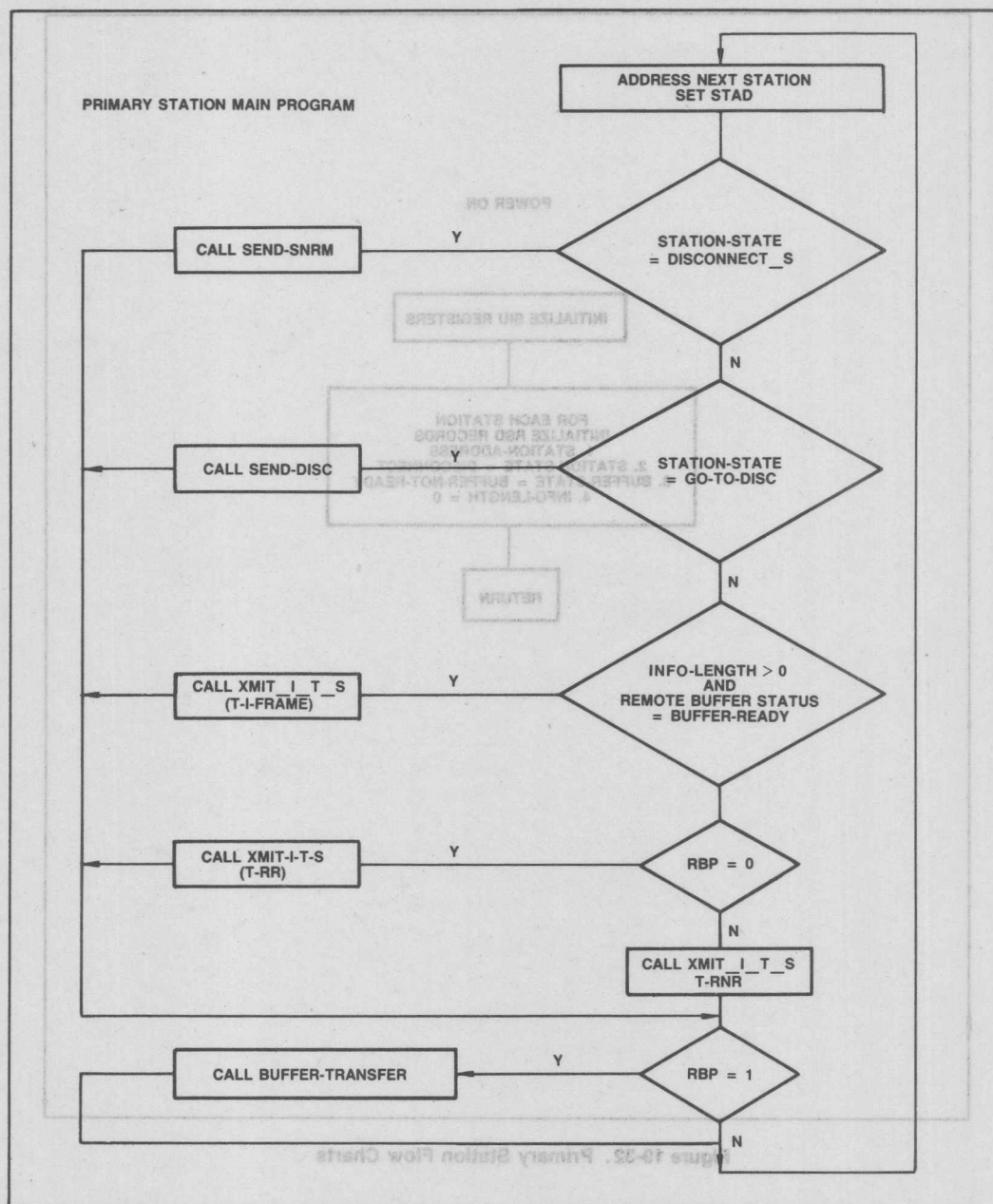


Figure 19-33. Primary Station Flow Charts



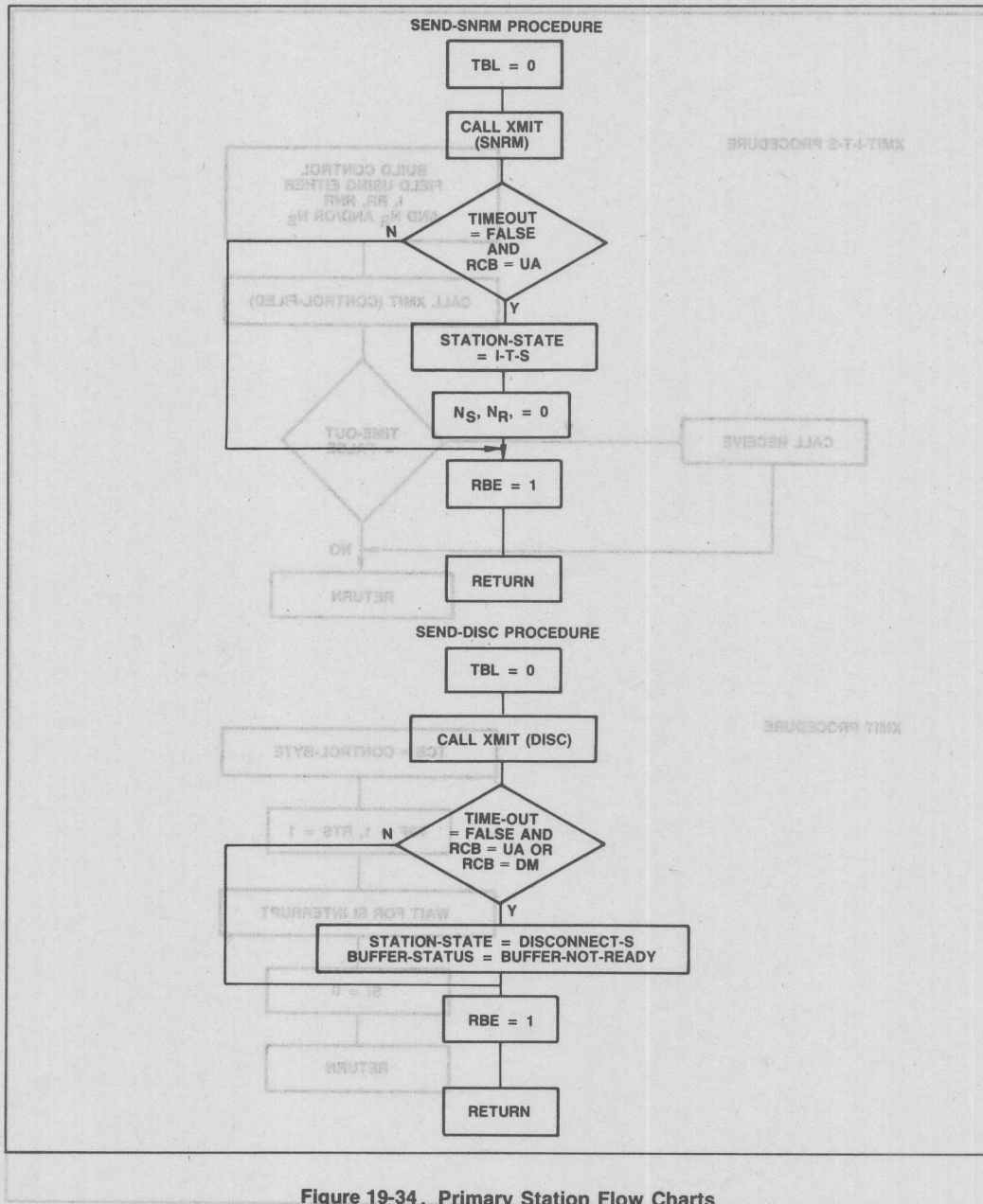


Figure 19-34. Primary Station Flow Charts

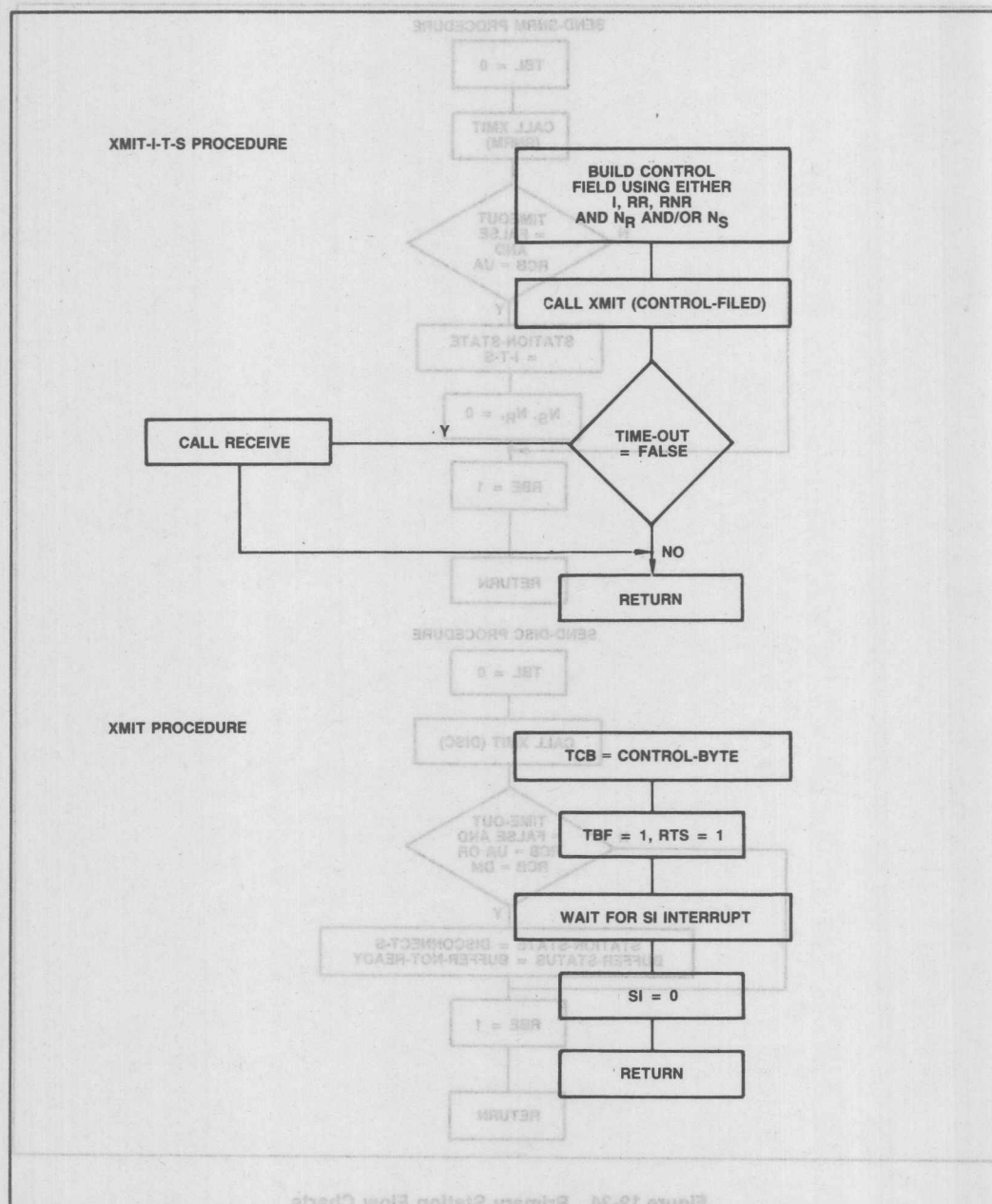


Figure 19-35. Primary Station Flow Charts

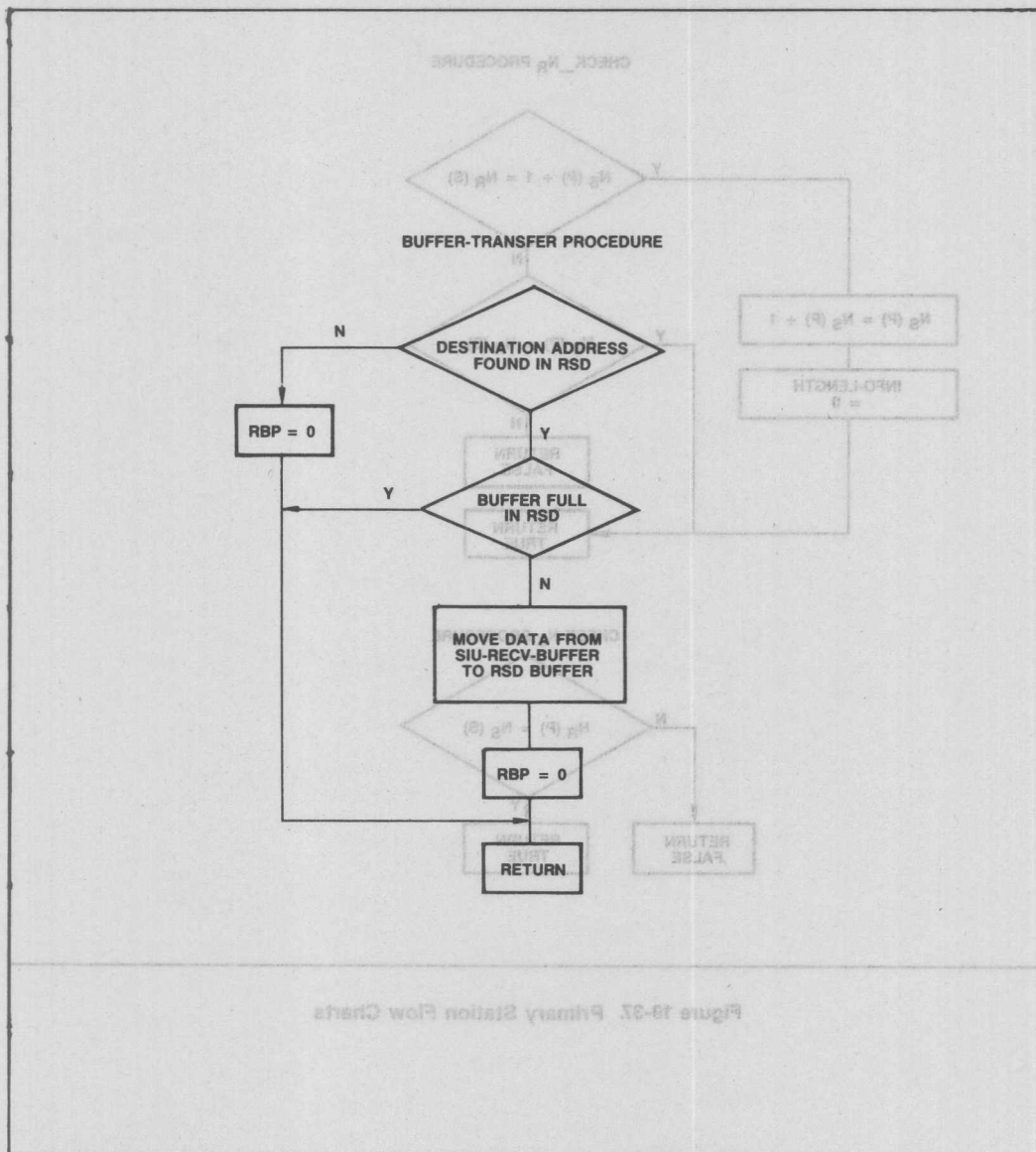


Figure 19-36. Primary Station Flow Charts

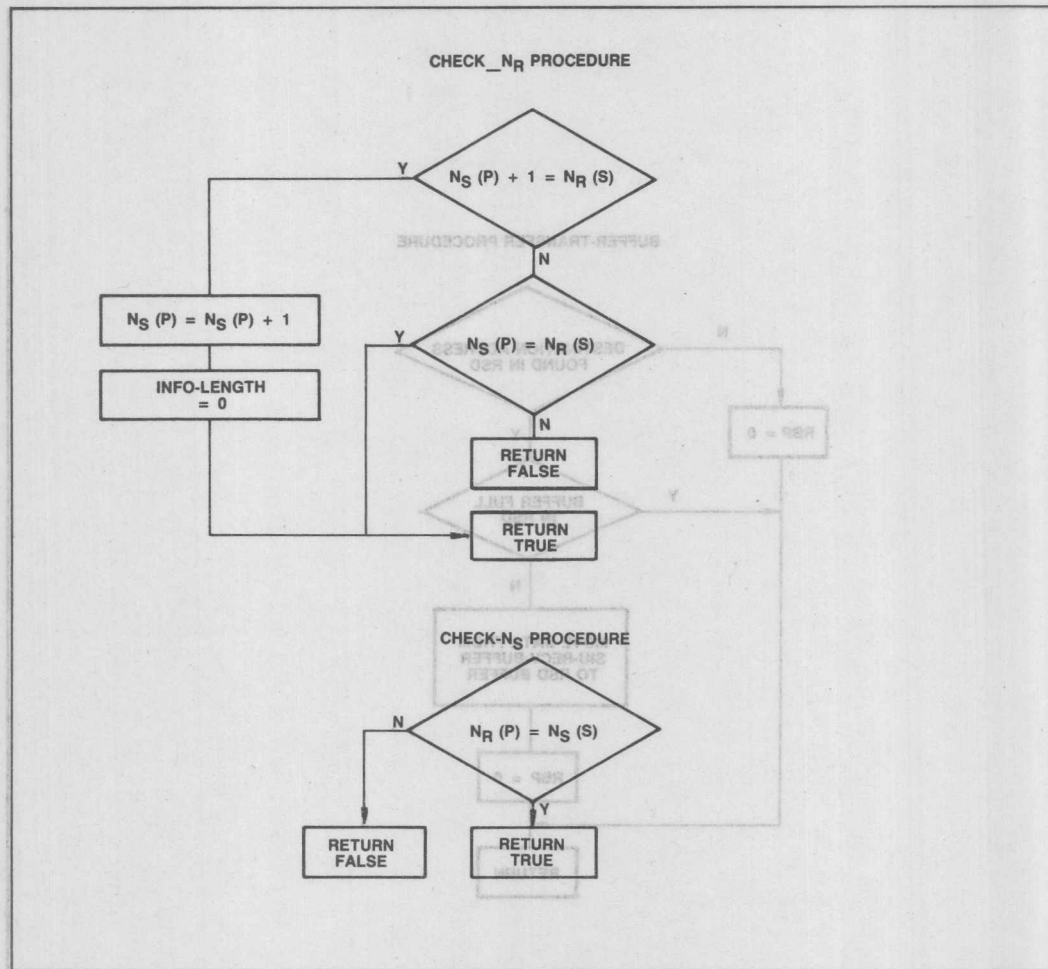


Figure 19-37. Primary Station Flow Charts

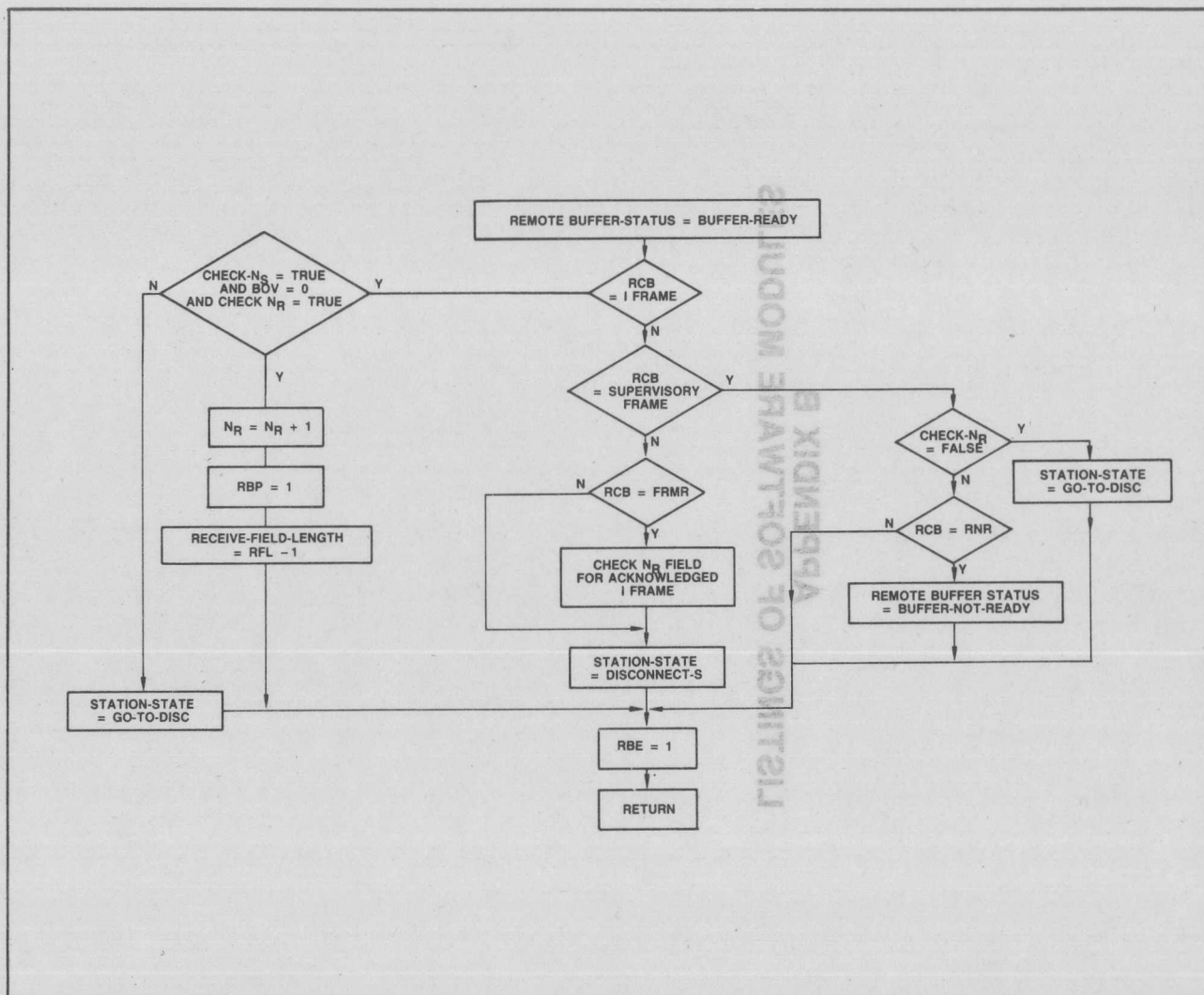
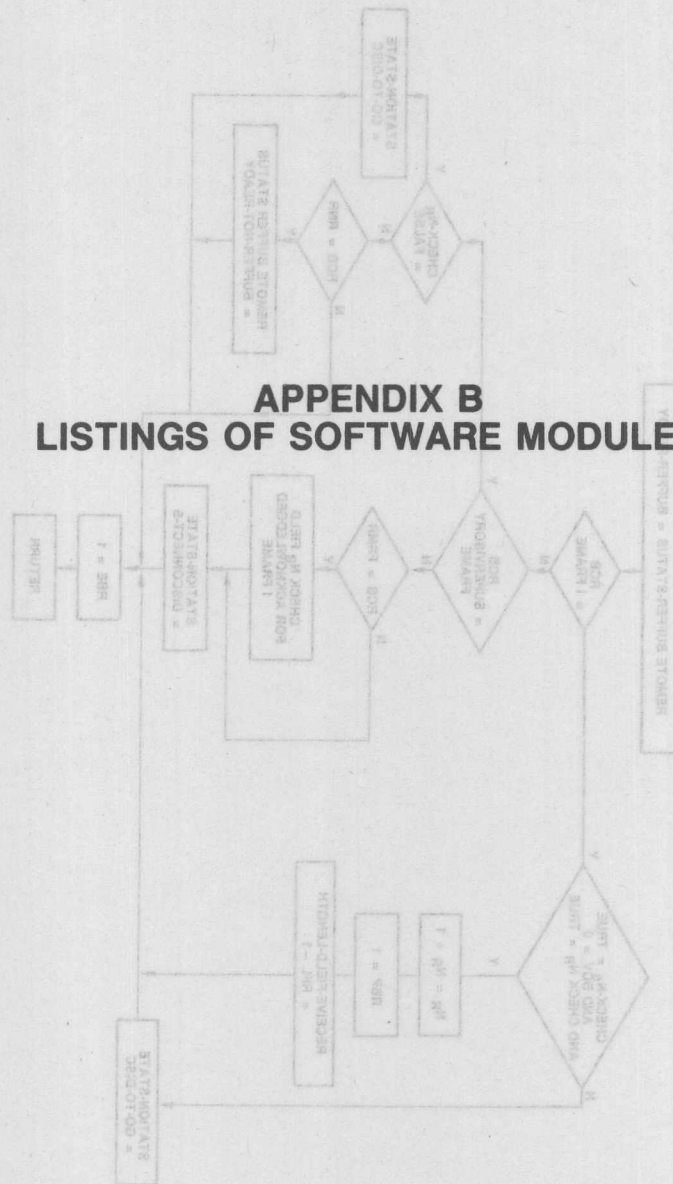


Figure 19-38. Primary Station Flow Charts



## APPENDIX B

### LISTINGS OF SOFTWARE MODULES



ISIS-II PL/M-51 V1.0

COMPILER INVOKED BY: :F2:PLM51 :F2:APNOTE.SRC

```

$TITLE      ('RUP1-44 Secondary Station Driver')
$DEBUG
$REGISTERBANK(1)
MAIN$MOD: DO;
$NOLIST

/* To save paper the RUP1 registers are not listed, but this is the statement
   used to include them: $INCLUDE (:F2:REQ44.DCL) */

5 1  DECLARE LIT      LITERALLY 'LITERALLY',
        TRUE      LIT      'OFFH',
        FALSE     LIT      'OOH',
        FOREVER    LIT      'WHILE 1',

/* SDLC commands and responses */

6 1  DECLARE SNRM      LIT      '83H',
        UA            LIT      '73H',
        DISC         LIT      '43H',
        DM           LIT      '1FH',
        FRMR         LIT      '97H',
        REQ_DISC     LIT      '53H',
        UP           LIT      '33H',
        TEST         LIT      '0E3H',

/* User states */
OPEN_S      LIT      'OOH',
CLOSED_S    LIT      '01H',

/* Station states */
DISCONNECT_S LIT      'OOH', /* LOGICALLY DISCONNECTED STATE */
FRMR_S       LIT      '01H', /* FRAME REJECT STATE */
I_T_S        LIT      '02H', /* INFORMATION TRANSFER STATE */

/* Status values returned from TRANSMIT procedure */
USER_STATE_CLOSED LIT      'OOH',
LINK_DISCONNECTED LIT      '01H',
OVERFLOW          LIT      '02H',
DATA_TRANSMITTED  LIT      '03H',

/* Parameters passed to XMIT_FRMR */
UNASSIGNED_C LIT      'OOH',
NO_I_FIELD_ALLOWED LIT    '01H',
BUFF_OVERRUN  LIT      '02H',
SES_ERR       LIT      '03H',

```

```

/* Variables */
USER_STATE      BYTE    AUXILIARY,
STATION_STATE   BYTE    AUXILIARY,
I_FRAME_LENGTH  BYTE    AUXILIARY,

/* Buffers */
BUFFER_LENGTH    LIT    '60',
SIU_XMIT_BUFFER(BUFFER_LENGTH)  BYTE    PUBLIC IDATA,
SIU_RECV_BUFFER(BUFFER_LENGTH)  BYTE    PUBLIC,
FRMR_BUFFER(3)   BYTE,

/* Flags */
XMIT_BUFFER_EMPTY  BIT PUBLIC,

7 2    SIU_RECV: PROCEDURE (LENGTH) EXTERNAL;
8 2    DECLARE LENGTH BYTE;
9 1    END SIU_RECV;

10 2   OPEN: PROCEDURE PUBLIC USING 2;
11 2   USER_STATE=OPEN_S;
12 1   END OPEN;

13 2   CLOSE: PROCEDURE PUBLIC USING 2;
14 2   AN=0;
15 2   USER_STATE=CLOSED_S;
16 1   END CLOSE;

17 2   POWER_ON_D: PROCEDURE PUBLIC USING 0;

18 2   USER_STATE=CLOSED_S;
19 2   STATION_STATE=DISCONNECT_S;
20 2   TBS= SIU_XMIT_BUFFER(0);
21 2   RBS= SIU_RECV_BUFFER(0);
22 2   RBL=BUFFER_LENGTH;
23 2   RBE=1; /* Enable the SIU's receiver */
24 2   XMIT_BUFFER_EMPTY=1;

25 1   END POWER_ON_D;

26 2   TRANSMIT: PROCEDURE (XMIT_BUFFER_LENGTH) BYTE PUBLIC USING 0;

/* User must check XMIT_BUFFER_EMPTY flag before calling this procedure */

27 2   DECLARE XMIT_BUFFER_LENGTH  BYTE,
I          BYTE    AUXILIARY,
STATUS     BYTE    AUXILIARY;

28 2   IF USER_STATE=CLOSED_S
THEN STATUS=USER_STATE_CLOSED;
30 2   ELSE IF STATION_STATE=DISCONNECT_S
THEN STATUS=LINK_DISCONNECTED;
32 2   ELSE IF XMIT_BUFFER_LENGTH>BUFFER_LENGTH
THEN STATUS=OVERFLOW;
34 3   ELSE DO;

```

```

35 3      XMIT_BUFFER_EMPTY=0;
36 3      TBL=XMIT_BUFFER_LENGTH;
37 3      I_FRAME_LENGTH=XMIT_BUFFER_LENGTH; /* Store length in case station
                                           is reset by FRMR, SNRM etc. */
38 3      TBF=1;
39 3      STATUS=DATA_TRANSMITTED;
40 3      END;
41 2      RETURN STATUS;
42 1      END TRANSMIT;

43 2      XMIT_UNNUMBERED: PROCEDURE (CONTROL_BYTE);
44 2          DECLARE CONTROL_BYTE    BYTE;
45 2          TCB=CONTROL_BYTE;
46 2          TBF=1;
47 2          RTS=1;
48 3          DO WHILE NOT SI;
49 3              END;
50 2          SI=0;

51 1      END XMIT_UNNUMBERED;

52 2      SNRM_RESPONSE: PROCEDURE ;
53 2          STATION_STATE=I_T_S;
54 2          NSNR=0;
55 2          IF (RCB AND IOH) <> 0 /* Respond if polled */
56 3              THEN DO;
57 3                  TBL=0;
58 3                  CALL XMIT_UNNUMBERED(UA);
59 3                  END;
60 2          IF XMIT_BUFFER_EMPTY=0 /* If an I frame was left pending transmission
                                   then restore it */
61 3              THEN DO;
62 3                  TBL=I_FRAME_LENGTH;
63 3                  TBF=1;
64 3                  END;
65 2          AN=1;

66 1      END SNRM_RESPONSE;

67 2      XMIT_FRMR: PROCEDURE (REASON);
68 2          DECLARE REASON    BYTE;
69 2          TCB=FRMR;
70 2          TBS= FRMR_BUFFER(0);
71 2          TBL=3;
72 2          FRMR_BUFFER(0)=RCB;
73 2          /* Swap nibbles in NSNR */
74 3          FRMR_BUFFER(1)=(SHL((NSNR AND OEH),4) OR SHR((NSNR AND OEH),4));
75 3          DO CASE REASON;
76 3              FRMR_BUFFER(2)=01H; /* UNASSIGNED_C */

```

```

76 3          FRMR_BUFFER(2)=02H; /* NO_I_FIELD_ALLOWED */
77 3          FRMR_BUFFER(2)=04H; /* BUFF_OVERRUN */
78 3          FRMR_BUFFER(2)=0BH; /* BES_ERR */
79 3      END;

80 2          STATION_STATE=FRMR_S;

81 2      IF (RCB AND 10H) <> 0
      THEN DO;
83 3          TBF=1;
84 3          RTS=1;
85 3          DO WHILE NOT SI;
86 3              END;
87 3          SI=0;
88 3      END;
89 1      END XMIT_FRMR;

90 2      IN_DISCONNECT_STATE: PROCEDURE /* Called from SIU_INT procedure */
91 2      IF ((USER_STATE=OPEN_S) AND ((RCB AND 0EFH)=SNRM))
      THEN CALL SNRM_RESPONSE;
93 2      ELSE IF (RCB AND 10H) <> 0
      THEN DO;
95 3          TBL=0;
96 3          CALL XMIT_UNNUMBERED(DM);
97 3      END;
98 1      END IN_DISCONNECT_STATE;

99 2      IN_FRMR_STATE: PROCEDURE /* Called by SIU_INT when a frame has been received
      when in the FRMR state */
100 2      IF (RCB AND 0EFH)=SNRM
      THEN DO;
102 3          CALL SNRM_RESPONSE;
103 3          TBS= SIU_XMIT_BUFFER(0); /* Restore transmit buffer start address */
104 3      END;

105 2      ELSE IF (RCB AND 0EFH)=DISC
      THEN DO;
107 3          STATION_STATE=DISCONNECT_S;
108 3          TBS= SIU_XMIT_BUFFER(0); /* Restore transmit buffer start address */
109 3          IF (RCB AND 10H) <> 0
          THEN DO;
111 4              TBL=0;
112 4              CALL XMIT_UNNUMBERED(UA);
113 4          END;
114 3      END;

115 3      ELSE DO; /* Receive control byte is something other than DISC or SNRM */
116 3      IF (RCB AND 10H) <> 0
      THEN DO;
118 4          TBF=1;
119 4          RTS=1;

```



```

120 5          DO WHILE NOT SI;
121 5          END;
122 4          END;
123 3          END;

124 1      END IN_FRMR_STATE;

125 2      COMMAND_DECODE: PROCEDURE ;

126 2          IF (RCB AND OEFH)=SNRM
              THEN CALL SNRM_RESPONSE;
128 2          ELSE IF (RCB AND OEFH)=DISC
              THEN DO;
130 3              STATION_STATE=DISCONNECT_S;
131 3              IF (RCB AND IOH)<0
                  THEN DO;
133 4                  TBL=0;
134 4                  CALL XMIT_UNNUMBERED(UA);
135 4                  END;
136 3              END;

137 2          ELSE IF (RCB AND OEFH)=TEST
              THEN DO;
139 3              IF (RCB AND IOH)>0 /* Respond if polled */
                  THEN DO; /* FOR BOV=1, SEND THE TEST RESPONSE WITHOUT AN I FIELD */
141 4                  IF (BOV=1) THEN DO;
143 5                      TBL=0;
144 5                      CALL XMIT_UNNUMBERED(TEST OR IOH);
145 5                      END;
146 5                      ELSE DO; /* If no BOV, send received I field back to primary */
147 5                          TBL=RFL;
148 5                          TBS=RBS;
149 5                          CALL XMIT_UNNUMBERED(TEST OR IOH);
150 5                          TBS=SIU_XMIT_BUFFER(O); /* Restore TBS */
151 5                          END;
                      /* If an I frame was pending, set it up again */
152 4                      IF XMIT_BUFFER_EMPTY=0
                          THEN DO;
154 5                          TBL=I_FRAME_LENGTH;
155 5                          TBF=1;
156 5                          END;
157 4                      END;
158 3                      AM=1;
159 3                      END;

160 2          ELSE IF (RCB AND OIH) = 0 /* Kicked out of the AUTO mode because
              an I frame was received while RPB = 1 */
              THEN DO;
162 3              AM = 1;
163 3              IF XMIT_BUFFER_EMPTY = 1
                  THEN TBL = 0;
165 3              TBF = 1; /* Send an AUTO mode response */

```

PL/M-51 COMPILER RUPI-44 Secondary Station Driver

20:24:47 09/20/83 PAGE 6

```

166 3          RTS = 1;
167 3          END;
168 2          ELSE CALL XMIT_FRMR(UNASSIGNED_C); /* Received an undefined or not implemented command */
169 1          END COMMAND_DECODE;

170 2          SIU_INT: PROCEDURE INTERRUPT 4;
171 2          DECLARE I BYTE AUXILIARY;
172 2          SI=0;
173 2          IF STATION_STATE<> I_T_S /* Must be in NON-AUTO mode */
174 2          THEN DO;
175 3              IF RBE=0 /* Received a frame? Give response */
176 3              THEN DO;
177 5                  DO CASE STATION_STATE;
178 5                  CALL IN_DISCONNECT_STATE;
179 5                  CALL IN_FRMR_STATE;
180 5                  END;
181 4                  RBE=1;
182 4              END;
183 3              RETURN;
184 3          END;

/* If the program reaches this point, STATION_STATE=I_T_S
which means the SIU either was, or still is in the AUTO MODE */

185 2          IF AM=0
186 2          THEN DO;
187 3              IF (RCB AND 0EFH)=DISC
188 3              THEN CALL COMMAND_DECODE;
189 3              ELSE IF USER_STATE=CLOSED_S
190 3              THEN DO;
191 4                  TBL=0;
192 4                  CALL XMIT_UNNUMBERED(REG_DISC);
193 4                  END;
194 3              ELSE IF SES=1
195 3              THEN CALL XMIT_FRMR(SES_ERR);
196 3              ELSE IF BOV=1
197 3              THEN DO; /* DON'T SEND FRMR IF A TEST WAS RECEIVED */
198 4                  IF (RCB AND 0EFH)=TEST
199 4                  THEN CALL COMMAND_DECODE;
200 4                  ELSE CALL XMIT_FRMR(BUFF_OVERRUN);
201 4                  END;
202 3              ELSE CALL COMMAND_DECODE;
203 3              RBE=1;
204 3              END;
205 3          ELSE DO; /* MUST STILL BE IN AUTO MODE */
206 3              IF TBF=0
207 3              THEN XMIT_BUFFER_EMPTY=1; /* TRANSMITTED A FRAME */
208 3              IF RBE=0
209 3              THEN DO;

```

PL/M-51 COMPILER RUPI-44 Secondary Station Driver

20:24:47 09/20/83 PAGE 7

```

210 4          RBP=1; /* RNR STATE */
211 4          RBE=1; /* RE-ENABLE RECEIVER */
212 4          CALL SIU_RECV(RFL);
213 4          RBP=0; /* RR STATE */
214 4          END;
215 3          END SIU_INT;
216 1          END MAIN$MOD;
217 1          END MAIN$MOD;

```

Software and application note written by Charles Yager

WARNINGS:  
4 IS THE HIGHEST USED INTERRUPT

| MODULE INFORMATION:     | (STATIC+OVERLAYABLE) |
|-------------------------|----------------------|
| CODE SIZE               | = 02BFH 655D         |
| CONSTANT SIZE           | = 0000H 0D           |
| DIRECT VARIABLE SIZE    | = 3FH+02H 63D+ 2D    |
| INDIRECT VARIABLE SIZE  | = 3CH+00H 60D+ 0D    |
| BIT SIZE                | = 01H+00H 1D+ 0D     |
| BIT-ADDRESSABLE SIZE    | = 00H+00H 0D+ 0D     |
| AUXILIARY VARIABLE SIZE | = 0006H 6D           |
| MAXIMUM STACK SIZE      | = 0017H 23D          |
| REGISTER-BANK(S) USED:  | 0 1 2                |
| 460 LINES READ          |                      |
| 0 PROGRAM ERROR(S)      |                      |

END OF PL/M-51 COMPILATION

PL/M-51 COMPILER Application Module: Async/SDLC Protocol converter 3.025 18:50:53 09/19/83 PAGE 1

IS18-II PL/M-51 V1.0

COMPILER INVOKED BY: :#2:plm51 :#2:unote.src

```

$TITLE      ('Application Module: Async/SDLC Protocol converter')
$debug
$registerbank(0)
user$mod:do;
$NOLIST
5 1  DECLARE      LIT      LITERALLY      'LITERALLY',
                  LIT      'OFFH',
                  LIT      '00H',
                  LIT      'WHILE 1',
                  LIT      '1BH',
                  LIT      '0AH',
                  LIT      '0DH',
                  LIT      '08H',
                  LIT      '07H',
                  LIT      '00H',
                  LIT      '01H',
                  LIT      '02H',
                  LIT      '00H',
                  LIT      '01H',
                  LIT      '02H',
                  LIT      '00H',
                  LIT      '01H',
                  LIT      '02H',
                  LIT      '03H',

/* BUFFERS */

BUFFER_LENGTH      LIT      '60',
SIU_XMIT_BUFFER(BUFFER_LENGTH)      BYTE      EXTERNAL      IDATA,
SIU_RECV_BUFFER(BUFFER_LENGTH)      BYTE      EXTERNAL,
FIFO_T(256)        BYTE      AUXILIARY,
IN_PTR_T           BYTE      AUXILIARY,
OUT_PTR_T          BYTE      AUXILIARY,
BUFFER_STATUS_T    BYTE      AUXILIARY,
FIFO_R(256)        BYTE      AUXILIARY,
IN_PTR_R           BYTE      AUXILIARY,
OUT_PTR_R          BYTE      AUXILIARY,
BUFFER_STATUS_R    BYTE      AUXILIARY,

/* Variables and Parameters */

LENGTH             BYTE      AUXILIARY,
CHAR               BYTE      AUXILIARY,
I                 BYTE      AUXILIARY,
USART_CMD          BYTE      AUXILIARY,
DESTINATION_ADDRESS      BYTE      AUXILIARY,
SEND_DATA          BYTE      AUXILIARY,
RESULT             BYTE      AUXILIARY,
ERR_MESSAGE_INDEX  BYTE      AUXILIARY,
ERR_MESSAGE_PTR    WORD      AUXILIARY,

/* Messages Sent to the Terminal */

PARITY(*) BYTE CONSTANT(LF,CR,'Parity Error Detected',LF,CR,00H),
FRAME(*)  BYTE CONSTANT(LF,CR,'Framing Error Detected',LF,CR,00H),

```

```

OVER_RUN(*) BYTE CONSTANT(LF,CR,'Overrun Error Detected',LF,CR,0),
LINK(*) BYTE CONSTANT(LF,CR,'Unable to Get Online',LF,CR,0),
DEST_ADDR(*) BYTE CONSTANT(CR,LF,LF,
    'Enter the destination address: ',BS,BS,0),

D_ADDR_ACK(*) BYTE CONSTANT(CR,LF,LF,
    'The new destination address is ',0),

STAT_ADDR(*) BYTE CONSTANT(CR,LF,LF,
    'Enter the station address: ',BS,BS,0),

S_ADDR_ACK(*) BYTE CONSTANT(CR,LF,LF,
    'The new station address is ',0),

ADDR_ACK_FIN(*) BYTE CONSTANT('H',CR,LF,LF,0),

SIGN_ON(*) BYTE CONSTANT(CR,LF,LF,
    '(\ /) RUPI-44 Secondary Station',CR,LF,
    ' \ /',CR,LF,LF,
    '1 - Set the Station Address',LF,CR,
    '2 - Set the Destination Address',CR,LF,
    '3 - Go Online',CR,LF,
    '4 - Go Offline',CR,LF,
    '5 - Return to terminal mode',CR,LF,LF,
    'Enter option: ',BS,0),

FIN(*) BYTE CONSTANT(CR,LF,LF,0),

/* Characters Received From the Terminal */
HEX_TABLE(17) BYTE CONSTANT('0123456789ABCDEF',BEL),
MENU_CHAR(6) BYTE CONSTANT('12345',BEL),

/* Flags and Bits */
XMIT_BUFFER_EMPTY BIT EXTERNAL, /* Semaphore for RUPI SIOU Transmit Buffer */
STOP_BIT BIT AT(147) REG, /* Terminal parameters */
ECHO BIT AT(0B4H) REG,
WAIT BIT, /* Timeout flag */
ERROR_FLAG BIT, /* Error message Flag */

/* Peripheral Addresses */
USART_STATUS BYTE AT(0B01H) AUXILIARY,
USART_DATA BYTE AT(0B00H) AUXILIARY,
TIMER_CONTROL BYTE AT(1003H) AUXILIARY,
TIMER_0 BYTE AT(1000H) AUXILIARY,
TIMER_1 BYTE AT(1001H) AUXILIARY,
TIMER_2 BYTE AT(1002H) AUXILIARY,

/* External Procedures */

```

```

6 2 POWER_ON_D: PROCEDURE EXTERNAL;
7 1 END POWER_ON_D;

8 2 CLOSE: PROCEDURE EXTERNAL USING 2;
9 1 END CLOSE;

10 2 OPEN: PROCEDURE EXTERNAL USING 2;
11 1 END OPEN;

12 2 TRANSMIT: PROCEDURE (XMIT_BUFFER_LENGTH) BYTE EXTERNAL;
13 2 DECLARE XMIT_BUFFER_LENGTH BYTE;
14 1 END TRANSMIT;

```

/\* Local Procedures \*/

```

15 2 TIMER_O_INT: PROCEDURE INTERRUPT 1 USING 1;
16 2 WAIT=0;
17 1 END TIMER_O_INT;

```

```

18 2 POWER_ON: PROCEDURE USING 0;

```

```

19 2 DECLARE TEMP BYTE AUXILIARY;

```

```

20 2 SMD=54H; /* Using DPLL, NRZI, PFS, TIMER 1, @ 52.5 Kbps */
21 2 THOD=21H; /* Timer 0 16 bit, Timer 1 auto reload */
22 2 TH1=0FFH;
23 2 TCON=40H;

```

```

24 2 TIMER_CONTROL=37H; /* Initialize USART's system clock; 8254 */
25 2 TIMER_O=04H;
26 2 TIMER_O=00H;
27 2 TIMER_CONTROL=77H; /* Initialize Tx, Rx */

```

/\* Definition for dip switch tied to P1.0 to P1.6

| Bit Rate | 3   | 2   | 1   |
|----------|-----|-----|-----|
| 300      | on  | on  | on  |
| 1200     | on  | on  | off |
| 2400     | on  | off | on  |
| 4800     | on  | off | off |
| 9600     | off | on  | on  |
| 19200    | off | on  | off |
| Stop bit | 4   |     |     |
| 1        | on  |     |     |
| 2        | off |     |     |
| Parity   | 6   | 5   |     |
| off      | on  | on  |     |
| odd      | on  | off |     |
| even     | off | on  |     |
|          | off | off |     |



```

Echo      7
on         on
off        off      */

28 2      TEMP=P1 AND 07H; /* Read the dip switch to determine the bit rate */
29 2      IF TEMP>5
30 3      THEN TEMP=0;
31 3      DO CASE TEMP;
32 4      /* 300 */ DO;
33 4      TIMER_1=83H;
34 4      TIMER_1=20H;
35 4      END;

36 4      /* 1200 */ DO;
37 4      TIMER_1=20H;
38 4      TIMER_1=05H;
39 4      END;

40 4      /* 2400 */ DO;
41 4      TIMER_1=60H;
42 4      TIMER_1=02H;
43 4      END;

44 4      /* 4800 */ DO;
45 4      TIMER_1=30H;
46 4      TIMER_1=01H;
47 4      END;

48 4      /* 9600 */ DO;
49 4      TIMER_1=65H;
50 4      TIMER_1=0;
51 4      END;

52 4      /* 19200 */ DO;
53 4      TIMER_1=33H;
54 4      TIMER_1=0;
55 4      END;
56 3      END;

57 2      USART_STATUS=0; /* Software power-on reset for 8251A */
58 2      USART_STATUS=0;
59 2      USART_STATUS=0;
60 2      USART_STATUS=40H;

61 2      TEMP=0AH; /* Determine the parity and # of stop bits */
62 2      TEMP=TEMP OR (P1 AND 30H);
63 2      IF STOP_BIT=1
64 3      THEN TEMP=TEMP OR 0COH;
65 2      ELSE TEMP=TEMP OR 40H;

66 2      USART_STATUS=TEMP; /* USART Mode Word */
67 2      USART_STATUS, USART_CMD=27H; /* USART Command Word RTS, RxE, DTR, TxEN=1 */
68 2      STAD=OFFH;

```

```

69 2      SEND_DATA=0; /* Initialize Flags */
70 2      IN_PTR_T, OUT_PTR_T, IN_PTR_R, OUT_PTR_R = 0; /* Initialize FIFO PTRs */
71 2      BUFFER_STATUS_T, BUFFER_STATUS_R= EMPTY;
72 2      CALL POWER_ON_D;
73 2      IP=01H; /* USART's RxRdy is the highest priority */
74 2      IE=93H; /* Both external interrupts are level triggered */
75 2      ERROR_FLAG=0; /* Enable USART RxRdy, SI, and Timer 0 interrupts */
76 1      END POWER_ON;
77 2      FIFO_R_IN: PROCEDURE (CHAR) USING 1;
78 2      DECLARE CHAR BYTE;
79 2      FIFO_R(IN_PTR_R)=CHAR;
80 2      IN_PTR_R=IN_PTR_R+1;
81 2      IF BUFFER_STATUS_R=EMPTY
82 3      THEN DO;
83 3          EA=0;
84 3          BUFFER_STATUS_R=INUSE;
85 3          EX1=1; /* Enable USART's TxD interrupt */
86 3          EA=1;
87 3      END;
88 2      ELSE IF ((BUFFER_STATUS_R=INUSE) AND (IN_PTR_R=OUT_PTR_R))
89 3      THEN BUFFER_STATUS_R=FULL;
90 1      END FIFO_R_IN;
91 2      FIFO_R_OUT: PROCEDURE BYTE USING 1;
92 2      DECLARE CHAR BYTE AUXILIARY;
93 2      CHAR=FIFO_R(OUT_PTR_R);
94 2      OUT_PTR_R=OUT_PTR_R+1;
95 2      IF OUT_PTR_R=IN_PTR_R
96 3      THEN DO;
97 3          EX1=0; /* Shut off TxD interrupt */
98 3          BUFFER_STATUS_R=EMPTY;
99 3      END;
100 2      ELSE IF ((BUFFER_STATUS_R=FULL) AND (OUT_PTR_R=20=IN_PTR_R))
101 3      THEN BUFFER_STATUS_R=INUSE;
102 2      RETURN CHAR;
103 1      END FIFO_R_OUT;
104 2      USART_XMIT_INT: PROCEDURE INTERRUPT 2 USING 1;

```

```

105 2      DECLARE
106           MESSAGE BASED ERR_MESSAGE_PTR(1)  BYTE  CONSTANT;
107
108 2      IF ERROR_FLAG
109           THEN DO;
110                IF MESSAGE(ERR_MESSAGE_INDEX) <> 0 /* Then continue to send the message */
111                THEN DO;
112                     USART_DATA = MESSAGE(ERR_MESSAGE_INDEX);
113                     ERR_MESSAGE_INDEX=ERR_MESSAGE_INDEX+1;
114                     END;
115
116                ELSE DO; /* If message is done reset ERROR_FLAG and shut off interrupt if FIFO is empty */
117                     ERROR_FLAG=0;
118                     IF BUFFER_STATUS_R = EMPTY
119                     THEN EX1=0;
120
121                     END;
122
123           ELSE USART_DATA=FIFO_R_OUT;
124
125 1      END USART_XMIT_INT;
126
127 2      SIU_RECV: PROCEDURE (LENGTH) PUBLIC USING 1;
128
129           DECLARE LENGTH  BYTE,
130                    I      BYTE  AUXILIARY;
131
132           DO I=0 TO LENGTH-1;
133               DO WHILE BUFFER_STATUS_R=FULL; /* Check to see if fifo is full */
134                   END;
135               CALL FIFO_R_IN(SIU_RECV_BUFFER(I));
136           END;
137
138 1      END SIU_RECV;
139
140 2      FIFO_T_IN: PROCEDURE (CHAR) USING 2;
141
142           DECLARE CHAR  BYTE;
143
144           FIFO_T(IN_PTR_T)=CHAR;
145           IN_PTR_T=IN_PTR_T+1;
146           IF CHAR=LF
147               THEN SEND_DATA=SEND_DATA+1;
148
149           IF BUFFER_STATUS_T=EMPTY
150               THEN BUFFER_STATUS_T=INUSE;
151           ELSE IF ((BUFFER_STATUS_T=INUSE) AND (IN_PTR_T+20=OUT_PTR_T))
152               THEN DO; /* Stop reception using CTS */
153                     USART_STATUS, USART_CMD=USART_CMD AND NOT(20H);
154                     BUFFER_STATUS_T=FULL;
155                     IF SEND_DATA=0
156                         THEN SEND_DATA=1; /* If the buffer is full and no LF
157                                           has been received then send data */
158
159                     END;
160
161 1      END FIFO_T_IN;

```

```

145 2      FIFO_T_OUT: PROCEDURE BYTE ;
146 2      DECLARE CHAR      BYTE      AUXILIARY;
147 2      CHAR=FIFO_T(OUT_PTR_T);
148 2      OUT_PTR_T=OUT_PTR_T+1;
149 2      IF OUT_PTR_T=IN_PTR_T      /* Then FIFO_T is empty */
      THEN DO;
151 3          EA=0;
152 3          BUFFER_STATUS_T=EMPTY;
153 3          SEND_DATA=0;
154 3          EA=1;
155 3      END;
156 2      ELSE IF ((BUFFER_STATUS_T=FULL) AND (OUT_PTR_T=IN_PTR_T))
      THEN DO;
158 3          USART_STATUS, USART_CMD=USART_CMD OR 20H;
159 3          BUFFER_STATUS_T=INUSE;
160 3      END;
161 2      IF (CHAR=LF AND SEND_DATA>0) THEN SEND_DATA=SEND_DATA-1;
162 2      RETURN CHAR;
163 2      END FIFO_T_OUT;
164 2
165 2      ERROR: PROCEDURE (STATUS) USING 2;
166 2      DECLARE STATUS      BYTE;
167 2      IF (STATUS AND 08H)<0
      THEN ERR_MESSAGE_PTR=. PARITY;
169 2      ELSE IF (STATUS AND 10H)<0
      THEN ERR_MESSAGE_PTR=. OVER_RUN;
171 2      ELSE IF (STATUS AND 20H)<0
      THEN ERR_MESSAGE_PTR=. FRAME;
173 2      USART_STATUS=(USART_CMD OR 10H); /* Reset error flags on USART */
174 2      ERR_MESSAGE_INDEX = 0;
175 2      ERROR_FLAG=1;
176 2      EX1=1; /* Turn on Tx Interrupt */
177 1      END ERROR;
178 2      LINK_DISC: PROCEDURE ;
      /* This procedure sends the message 'Unable to Get Online' to the terminal */
179 2      DECLARE MESSAGE_PTR WORD      AUXILIARY,
      MESSAGE      BASED      MESSAGE_PTR(1) CHAR BYTE CONSTANT,
      J      BYTE      AUXILIARY,
      EX1_STORE BIT;
180 2      EX1_STORE=EX1; /* Shut off async transmit interrupt */
181 2      EX1=0;
182 2      MESSAGE_PTR=. LINK;
183 2      J=0;
184 3      DO WHILE (MESSAGE(J)<0);

```

```

185 4      DO WHILE (USART_STATUS AND 01H)=0; /* Wait for TxRDY on USART */
186 4      END;
187 3      USART_DATA=MESSAGE(J);
188 3      J=J+1;
189 3      END;
190 2      EX1=EX1_STORE; /* Restore async transmit interrupt */
191 1      END LINK_DISC;

192 2      CO: PROCEDURE (CHAR) USING 2;
193 2      DECLARE CHAR BYTE;
194 3      DO WHILE (USART_STATUS AND 01H) = 0;
195 3      END;
196 2      USART_DATA=CHAR;
197 1      END CO;
198 2      CI: PROCEDURE BYTE USING 2;
199 3      DO WHILE (USART_STATUS AND 02H) = 0;
200 3      END;
201 2      RETURN USART_DATA;
202 1      END CI;

203 2      GET_HEX: PROCEDURE BYTE USING 2;
204 2      DECLARE CHAR BYTE AUXILIARY,
                I BYTE AUXILIARY;
205 2      LO: CHAR=CI;
206 3      DO I=0 TO 15;
207 3      IF CHAR=HEX_TABLE(I)
                THEN GOTO L1;
208 3      END;
209 3      L1: CALL CO(HEX_TABLE(I));
210 2      IF I=16
211 2      THEN GOTO LO;
212 2      RETURN I;
213 2      END GET_HEX;
214 1      END GET_HEX;
215 2      OUTPUT_MESSAGE: PROCEDURE (MESSAGE_PTR) USING 2;
216 2      DECLARE MESSAGE_PTR WORD,
                MESSAGE BASED MESSAGE_PTR(1) BYTE CONSTANT,
                I BYTE AUXILIARY;
217 2      I=0;
218 3      DO WHILE MESSAGE(I) <> 0;
219 3      CALL CO(MESSAGE(I));
220 3      I=I+1;

```



```

221 3      END;
222 1      END OUTPUT_MESSAGE;
223 2      MENU: PROCEDURE USING 2;
224 2      DECLARE I          BYTE    AUXILIARY,
                CHAR        BYTE    AUXILIARY,
                STATION_ADDRESS BYTE  AUXILIARY;
225 2      START:
                CALL OUTPUT_MESSAGE(.SIGN_ON);
226 2      MO: CHAR=CI;      /* Read a character */
227 3      DO I=0 TO 4;
228 3      IF CHAR=MENU_CHAR(I)
                THEN GOTO M1;
230 3      END;
231 2      M1: CALL CO(MENU_CHAR(I));
232 2      IF I=5
                THEN GOTO MO;
234 3      DO CASE I;
235 4      DO;
236 4      CALL OUTPUT_MESSAGE(.STAT_ADDR);
237 4      STATION_ADDRESS=SHL(GET_HEX, 4);
238 4      STATION_ADDRESS=(STATION_ADDRESS OR GET_HEX);
239 4      STAD=STATION_ADDRESS;
240 4      CALL OUTPUT_MESSAGE(.S_ADDR_ACK);
241 4      CALL CO(HEX_TABLE(SHL(STATION_ADDRESS, 4)));
242 4      CALL CO(HEX_TABLE(OFH AND STATION_ADDRESS));
243 4      CALL OUTPUT_MESSAGE(.ADDR_ACK_FIN);
244 4      END;
245 4      DO;
246 4      CALL OUTPUT_MESSAGE(.DEST_ADDR);
247 4      DESTINATION_ADDRESS=SHL(GET_HEX, 4);
248 4      DESTINATION_ADDRESS=(DESTINATION_ADDRESS OR GET_HEX);
249 4      CALL OUTPUT_MESSAGE(.D_ADDR_ACK);

```

```

250 4      CALL CO(HEX_TABLE(SHR(DESTINATION_ADDRESS,4)));
251 4      CALL CO(HEX_TABLE(OFH AND DESTINATION_ADDRESS));
252 4      CALL OUTPUT_MESSAGE(.ADDR_ACK_FIN);
253 4      END;
254 4      DO;
255 4          CALL OUTPUT_MESSAGE(.FIN);
256 4          CALL OPEN;
257 4      END;
258 4      DO;
259 4          CALL OUTPUT_MESSAGE(.FIN);
260 4          CALL CLOSE;
261 4      END;
262 3      CALL OUTPUT_MESSAGE(.FIN);
263 3      END; /* DO CASE */
264 1      END MENU;
265 2      USART_RECV_INT: PROCEDURE INTERRUPT 0 USING 2;
266 2          DECLARE CHAR      BYTE      AUXILIARY,
                STATUS      BYTE      AUXILIARY;
267 2          CHAR=USART_DATA;
268 2          STATUS=USART_STATUS AND 3BH;
269 2          IF STATUS<>0
                THEN CALL ERROR(STATUS);
271 2          ELSE IF CHAR=ESC
                THEN CALL MENU;
273 3          ELSE DO;
274 3              CALL FIFO_T_IN(CHAR);
275 3              IF ECHO=0
                THEN CALL CO(CHAR);
277 3          END;
278 1      END USART_RECV_INT;
279 1      BEGIN;
                CALL POWER_ON;
280 2          DO FOREVER;
281 2              IF SEND_DATA<0
                THEN DO;
283 4                  DO WHILE NOT(XMIT_BUFFER_EMPTY); /*Wait until SIU_XMIT_BUFFER
                                                                is empty */
284 4                      END;
285 3                      LENGTH, CHAR =1;
286 3                      SIU_XMIT_BUFFER(0)=DESTINATION_ADDRESS;
287 4                      DO WHILE ((CHAR<>LF) AND (LENGTH<BUFFER_LENGTH) AND (BUFFER_STATUS_T<>EMPTY));

```

PL/M-51 COMPILER Application Module: Async/SDLC Protocol converter

18:50:53 09/19/83 PAGE 111

```

288 4          CHAR=FIFO_T_OUT;
289 4          SIU_XMIT_BUFFER(LENGTH)=CHAR;
290 4          LENGTH=LENGTH+1;
291 4          END;

/* If the line entered at the terminal is greater than BUFFER_LENGTH char. send the
first BUFFER_LENGTH char. then send the rest; since the SIU buffer is only BUFFER_LENGTH bytes */
292 3  L1:      I=0; /* Use I to count the number of unsuccessful
                transmits */

293 3  RETRY:    RESULT=TRANSMIT(LENGTH); /* Send the message */
294 3          IF RESULT<>DATA_TRANSMITTED
                THEN DO;
                /* Wait 50 msec for link to connect then try again */
296 4          WAIT=1;
297 4          TH0=3CH;
298 4          TLO=0AFH;
299 4          TRO=1;
300 5          DO WHILE WAIT;
301 5          END;
302 4          TRO=0;
303 4          I=I+1;
304 5          IF I>100 THEN DO; /* Wait 5 sec to get on line else
                                send error message to terminal
                                and try again */
                                CALL LINK_DISC;
                                GOTO L1;
                                END;
                                GOTO RETRY;
                                END;
                                END;
311 3          END;

312 2          END;
313 1          END USER*MOD;

```

WARNINGS:  
2 IS THE HIGHEST USED INTERRUPT

MODULE INFORMATION:  
CODE SIZE  
CONSTANT SIZE  
DIRECT VARIABLE SIZE  
INDIRECT VARIABLE SIZE  
BIT SIZE  
BIT-ADDRESSABLE SIZE  
AUXILIARY VARIABLE SIZE  
MAXIMUM STACK SIZE  
REGISTER-BANK(S) USED:  
713 LINES READ  
0 PROGRAM ERROR(S)  
END OF PL/M-51 COMPILATION

(STATIC+OVERLAYABLE)  
= 0682H 1714D  
= 01CFH 463D  
= 00H+05H 0D+ 5D  
= 00H+00H 0D+ 0D  
= 02H+01H 2D+ 1D  
= 00H+00H 0D+ 0D  
= 021FH 543D  
= 0028H 40D  
0 1 2

ISIS-II PL/M-51 V1.0

COMPILER INVOKED BY: :F2:PLM51 :F2:PNOTE.SRC

```

*TITLE 'RUP1-44 Primary Station'
*DEBU0
*REGISTERBANK(0)
MAIN$MOD:DD;

/* To save paper the RUP1 registers are not listed, but this is the statement
   used to include them: *INCLUDE (:F2:REQ44.DCL);*/

*ENLIST

5 1 DECLARE LIT LITERALLY 'LITERALLY',
      TRUE LIT 'OFFH',
      FALSE LIT 'OOH',
      FOREVER LIT 'WHILE 1';

/* SDLC COMMANDS AND RESPONSES */

6 1 DECLARE SNRM LIT '93H',
      UA LIT '73H',
      DISC LIT '53H',
      DH LIT '1FH',
      FRMR LIT '97H',
      REQ_DISC LIT '53H',
      UP LIT '33H',
      TEST LIT 'OF3H',
      RR LIT '11H',
      RNR LIT '15H',

```

/\* REMOTE STATION BUFFER STATUS \*/

```

BUFFER_READY LIT '0',
BUFFER_NOT_READY LIT '1',

```

/\* STATION STATES \*/

```

DISCONNECT_S LIT 'OOH', /* LOGICALLY DISCONNECTED STATE */
OO_TO_DISC LIT '01H',
I_T_S LIT '02H', /* INFORMATION TRANSFER STATE */

```

/\* PARAMETERS PASSED TO XMIT\_I\_T\_S \*/

```

T_I_FRAME LIT 'OOH',
T_RR LIT '01H',
T_RNR LIT '02H',

```

/\* SECONDARY STATION IDENTIFICATION \*/

```

NUMBER_OF_STATIONS LIT '2',
SECONDARY_ADDRESSES(NUMBER_OF_STATIONS)
      BYTE CONSTANT(55H, 43H),

```

```

/* Remote Station Database */

RSD(NUMBER_OF_STATIONS) STRUCTURE
(STATION_ADDRESS BYTE,
 STATION_STATE   BYTE,
 NS              BYTE,
 NR              BYTE,
 BUFFER_STATUS   BYTE, /* The status of the secondary stations buffer */
 INFO_LENGTH     BYTE,
 DATA(64)       BYTE)  AUXILIARY,

/* VARIABLES */
STATION_NUMBER  BYTE, /* AUXILIARY,
RECV_FIELD_LENGTH BYTE, /* AUXILIARY,
WAIT           BIT,

/* BUFFERS */
SIU_XMIT_BUFFER(64) BYTE IDATA,
SIU_RECV_BUFFER(64)  BYTE;

7 2  POWER_ON: PROCEDURE ;
8 2  DECLARE I BYTE AUXILIARY;
9 2  TBS= SIU_XMIT_BUFFER(0);
10 2 RBS= SIU_RECV_BUFFER(0);
11 2 RBL=64; /* 64 Byte receive buffer */
12 2 RBE=1; /* Enable the SIU's receiver */
13 3 DO I= 0 TO NUMBER_OF_STATIONS-1;
14 3 RSD(I). STATION_ADDRESS=SECONDARY_ADDRESSES(I);
15 3 RSD(I). STATION_STATE=DISCONNECT_S;
16 3 RSD(I). BUFFER_STATUS=BUFFER_NOT_READY;
17 3 RSD(I). INFO_LENGTH=0;
18 3 END;
19 2 SMD=54H; /* Using DLL, NRZI, PFS, TIMER 1, @ 62.5 Kbps */
20 2 THOD=21H;
21 2 TH1=OFFH;
22 2 TCON=40H; /* Use timer 0 for receive time out interrupt */
23 2 IE=82H;
24 1 END POWER_ON;
25 2 XMIT: PROCEDURE (CONTROL_BYTE);
26 2 DECLARE CONTROL_BYTE BYTE;
27 2 TCB=CONTROL_BYTE;
28 2 TBF=1;

```



```

29 2      RTS=1;
30 3      DO WHILE NOT SI;
31 3      END;
32 2      SI=0;

33 1      END XMIT;

34 2      TIMER_0_INT: PROCEDURE INTERRUPT 1; USING 1;
35 2      WAIT=0;
36 1      END TIMER_0_INT;

37 2      TIME_OUT: PROCEDURE BYTE;
/* Time_out returns true if there wasn't
a frame received within 200 msec.
If there was a frame received within
200 msec then time_out returns false. */

38 2      DECLARE I BYTE AUXILIARY;
39 3      DO I=0 TO 3;
40 3      WAIT=1;
41 3      THO=3CH;
42 3      TLO=0AFH;
43 3      TRO=1;
44 4      DO WHILE WAIT;
45 4      IF SI=1
THEN GOTO T_01;
47 4      END;
48 3      END;
49 2      RETURN TRUE;

50 2      T_01:
SI=0;
RETURN FALSE;

51 2      END TIME_OUT;

52 1      SEND_DISC: PROCEDURE;
53 2      TBL=0;
CALL XMIT(DISC);
IF TIME_OUT=FALSE
THEN IF RCB=UA OR RCB=DM
THEN DO;
57 3      RSD(STATION_NUMBER). BUFFER_STATUS=BUFFER_NOT_READY;
58 3      RSD(STATION_NUMBER). STATION_STATE=DISCONNECT_S;
59 3      END;
60 3      RBE=1;
61 3      END;
62 2      END SEND_DISC;

63 1      SEND_SNRM: PROCEDURE;
64 2      TBL=0;

```

```

66 2      CALL XMIT(SNRM);
67 2      IF (TIME_OUT=FALSE) AND (RCB=UA)
          THEN DO;
69 3          RSD(STATION_NUMBER).STATION_STATE=I_T_S;
70 3          RSD(STATION_NUMBER).NS=0;
71 3          RSD(STATION_NUMBER).NR=0;
72 3      END;
73 2      RBE=1;

74 1      END SEND_SNRM;

75 2      CHECK_NS: PROCEDURE BYTE;

          /* Check the Ns Field of the received frame. If Nr(P)=Ns(S) return true */
76 2      IF (RSD(STATION_NUMBER).NR=(SHR(RCB,1) AND 07H)
          THEN RETURN TRUE;
78 2      ELSE RETURN FALSE;

79 1      END CHECK_NS;

80 2      CHECK_NR: PROCEDURE BYTE;

          /* Check the Nr field of the received frame. If Ns(P)+1=Nr(S) then the frame
          has been acknowledged, else if Ns(P)=Nr(S) then the frame has not been
          acknowledged, else reset the secondary */
81 2      IF (((RSD(STATION_NUMBER).NS + 1) AND 07H) = SHR(RCB,5))
          THEN DO;
83 3          RSD(STATION_NUMBER).NS=((RSD(STATION_NUMBER).NS+1) AND 07H);
84 3          RSD(STATION_NUMBER).INFO_LENGTH=0;
85 3      END;
86 2      ELSE IF (RSD(STATION_NUMBER).NS <> SHR(RCB,5))
          THEN RETURN FALSE;

88 2      RETURN TRUE;

89 1      END CHECK_NR;

90 2      RECEIVE: PROCEDURE ;

91 2      DECLARE I BYTE AUXILIARY;

92 2      RSD(STATION_NUMBER).BUFFER_STATUS=BUFFER_READY;

          /* If an RNR was received buffer_status will be changed in the supervisory
          frame decode section futher down in this procedure, any other response
          means the remote stations buffer is ready */

93 2      IF (RCB AND 01H)=0
          THEN DO; /* 1 Frame Received */
95 3      IF (CHECK_NS=TRUE AND BOV=0 AND CHECK_NR=TRUE)
          THEN DO;
97 4          RSD(STATION_NUMBER).NR=((RSD(STATION_NUMBER).NR+1) AND 07H);
98 4          RBP=1;

```

```

99 4          RECV_FIELD_LENGTH=RFL-1;
100 4          END;
101 3          ELSE RSD(STATION_NUMBER).STATION_STATE=GO_TO_DISC;
102 3          END;
103 2          ELSE IF (RCB AND 03H)=01H
105 3          THEN DO; /* Supervisory frame received */
106 3          IF CHECK_NR=FALSE
107 3          THEN RSD(STATION_NUMBER).STATION_STATE=GO_TO_DISC;
108 3          ELSE IF ((RCB AND 0FH)=05H) /* then RNR */
109 3          THEN RSD(STATION_NUMBER).BUFFER_STATUS=BUFFER_NOT_READY;
110 3          END;
111 3          ELSE DO; /* Unnumbered frame or unknown frame received */
112 3          IF RCB=FRMR
113 3          THEN DO; /* If FRMR was received check Nr for an
114 3          acknowledged I frame */
115 3          RCB=SIU_RECV_BUFFER(1);
116 3          I=CHECK_NR;
117 3          END;
118 3          RSD(STATION_NUMBER).STATION_STATE=GO_TO_DISC;
119 3          END;
120 2          RBE=1;
121 1          END RECEIVE;
122 2          XMIT_I_T_S: PROCEDURE (TEMP);
123 2          DECLARE TEMP BYTE;
124 2          IF TEMP=T_I_FRAME
125 2          THEN DO; /* Transmit I frame */
126 2          /* Transfer the station buffer into internal ram */
127 2          DO TEMP=0 TO RSD(STATION_NUMBER).INFO_LENGTH-1;
128 2          SIU_XMIT_BUFFER(TEMP)=RSD(STATION_NUMBER).DATA(TEMP);
129 2          END;
130 2          /* Build the I frame control field */
131 2          TEMP=(SHL(RSD(STATION_NUMBER).NR,5) OR SHL(RSD(STATION_NUMBER).NS,1) OR 10H);
132 2          TBL=RSD(STATION_NUMBER).INFO_LENGTH;
133 2          CALL XMIT(TEMP);
134 2          IF TIME_OUT=FALSE
135 2          THEN CALL RECEIVE;
136 2          END;
137 3          ELSE DO; /* Transmit RR or RNR */
138 3          IF TEMP=T_RR
139 3          THEN TEMP=RR;
140 3          ELSE TEMP=RNR;
141 3          END;
142 3          CALL XMIT(TEMP);
143 3          END;

```

```

137 3          TEMP=(SHL(RSD(STATION_NUMBER),NR,5) OR TEMP);
138 3          TBL=0;
139 3          CALL XMIT(TEMP);
140 3          IF TIME_OUT=FALSE
              THEN CALL RECEIVE;
142 3          END;
143 1      END XMIT_I_T_S;
144 2      BUFFER_TRANSFER: PROCEDURE;
145 2          DECLARE I      BYTE    AUXILIARY,
                     J      BYTE    AUXILIARY;

146 3          DO I=0 TO NUMBER_OF_STATIONS-1;
147 3              IF RSD(I).STATION_ADDRESS=SIU_RECV_BUFFER(0)
                  THEN GOTO T1;
149 3          END;
150 2          T1:  IF I=NUMBER_OF_STATIONS /* If the addressed station does not exists,
              then discard the data */
              THEN DO;
152 3                  RBP=0;
153 3                  RETURN;
154 3              END;
155 2          ELSE IF RSD(I).INFO_LENGTH=0
              THEN DO;
157 3                  RSD(I).INFO_LENGTH=RECV_FIELD_LENGTH;
158 4                  DO J=1 TO RECV_FIELD_LENGTH;
159 4                      RSD(I).DATA(J-1)=SIU_RECV_BUFFER(J);
160 4                  END;
161 3                  RBP=0;
162 3              END;
163 1      END BUFFER_TRANSFER;

164 1      BEGIN;
          CALL POWER_ON;
165 2          DO FOREVER;
166 3              DO STATION_NUMBER=0 TO NUMBER_OF_STATIONS-1;
167 3                  STAD=RSD(STATION_NUMBER).STATION_ADDRESS;
168 3                  IF RSD(STATION_NUMBER).STATION_STATE = DISCONNECT_S
                      THEN CALL SEND_SNRM;
170 3                  ELSE IF RSD(STATION_NUMBER).STATION_STATE = GO_TO_DISC
                      THEN CALL SEND_DISC;
172 3                  ELSE IF ((RSD(STATION_NUMBER).INFO_LENGTH>0) AND
                      (RSD(STATION_NUMBER).BUFFER_STATUS=BUFFER_READY))
                      THEN CALL XMIT_I_T_S(T_I_FRAME);
174 3                  ELSE IF RBP=0
                      THEN CALL XMIT_I_T_S(T_RR);
176 3                  ELSE CALL XMIT_I_T_S(T_RNR);
177 3                  IF RBP=1
                      THEN CALL BUFFER_TRANSFER;

```

```

179 3          END;
180 2          END;
181 1      END MAIN$MOD;

```

WARNINGS:  
1 IS THE HIGHEST USED INTERRUPT

|                            |                      |
|----------------------------|----------------------|
| MODULE INFORMATION:        | (STATIC+OVERLAYABLE) |
| CODE SIZE                  | = 053DH 1341D        |
| CONSTANT SIZE              | = 0002H 2D           |
| DIRECT VARIABLE SIZE       | = 40H+02H 64D+ 2D    |
| INDIRECT VARIABLE SIZE     | = 40H+00H 64D+ 0D    |
| BIT SIZE                   | = 01H+00H 1D+ 0D     |
| BIT-ADDRESSABLE SIZE       | = 00H+00H 0D+ 0D     |
| AUXILIARY VARIABLE SIZE    | = 0093H 147D         |
| MAXIMUM STACK SIZE         | = 0019H 25D          |
| REGISTER-BANK(S) USED:     | 0 1                  |
| 456 LINES READ             |                      |
| 0 PROGRAM ERROR(S)         |                      |
| END OF PL/M-51 COMPILATION |                      |









# 8044AH/8344AH/8744H HIGH PERFORMANCE 8-BIT MICROCONTROLLER WITH ON-CHIP SERIAL COMMUNICATION CONTROLLER

- 8044AH — Includes Factory Mask Programmable ROM
- 8344AH — For Use With External Program Memory
- 8744H — Includes User Programmable/Erasable EPROM

## 8051 MICROCONTROLLER CORE

- Optimized for Real Time Control
  - 12 MHz Clock, Priority Interrupts,
  - 32 Programmable I/O lines,
  - Two 16-bit Timer/Counters
- Boolean Processor
- 4K x 8 ROM, 192 x 8 RAM
- 64K Accessible External Program Memory
- 64K Accessible External Data Memory
- 4  $\mu$ s Multiply and Divide

The RUP1-44 family integrates a high performance 8-bit Microcontroller, the Intel 8051 Core, with an intelligent/high performance HDLC/SDLC serial communication controller, called the Serial Interface Unit (SIU). See Figure 1. This dual architecture allows complex control and high speed data communication functions to be realized cost effectively.

Specifically, the 8044's Microcontroller features: 4K byte On-Chip program memory space; 32 I/O lines; two 16-bit timer/event counters; a 5-source; 2-level interrupt structure; a full duplex serial channel; a Boolean processor; and on-chip oscillator and clock circuitry. Standard TTL and most byte-oriented MCS-80 and MCS-85 peripherals can be used for I/O and memory expansion.

The Serial Interface Unit (SIU) manages the interface to a high speed serial link. The SIU offloads the On-Chip 8051 Microcontroller of communication tasks, thereby freeing the CPU to concentrate on real time control tasks.

The RUP1-44 family consists of the 8044, 8744, and 8344. All three devices are identical except in respect of on-chip program memory. The 8044 contains 4K bytes of mask-programmable ROM. User program-programmable EPROM replaces ROM in the 8744. The 8344 addresses all program memory externally.

The RUP1-44 devices are fabricated with Intel's reliable +5 volt, silicon-gate HMOSII technology and packaged in a 40-pin DIP.

The 8744H is available in a hermetically sealed, ceramic, 40-lead dual in-line package which includes a window that allows for EPROM erasure when exposed to ultraviolet light (See Erasure Characteristics). During normal operation, ambient light may adversely affect the functionality of the chip. Therefore, applications which expose the 8744H to ambient light may require an opaque label over the window.

## SERIAL INTERFACE UNIT (SIU)

- Serial Communication Processor that Operates Concurrently to CPU
- 2.4 Mbps Maximum Data Rate
- 375 Kbps using On-Chip Phase Locked Loop
- Communication Software in Silicon:
  - Complete Data Link Functions
  - Automatic Station Responses
- Operates as an SDLC Primary or Secondary Station

## 8044's Dual Controller Architecture

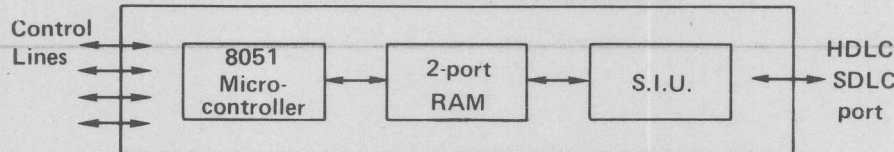
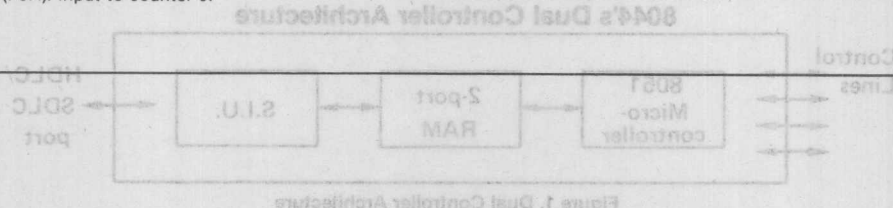


Figure 1. Dual Controller Architecture

Table 1. RUPI™-44 Family Pin Description

|   |   |
|---|---|
| <b>VSS</b><br>Circuit ground potential.   | —SCLK T1 (P3.5). In addition to I/O, this pin provides input to counter 1 or serves as SCLK (serial clock) input.   |
| <b>VCC</b><br>+5V power supply during operation and program verification.   | —WR (P3.6). The write control signal latches the data byte from Port 0 into the External Data Memory.   |
|   | —RD (P3.7). The read control signal enables External Data Memory to Port 0.   |
| <b>PORT 0</b><br>Port 0 is an 8-bit open drain bidirectional I/O port. It is also the multiplexed-low-order address and data bus when using external memory. It is used for data output during program verification. Port 0 can sink/source eight LS TTL loads.   | <b>RST</b><br>A high on this pin for two machine cycles while the oscillator is running resets the device. A small external pulldown resistor ( $\approx 8.2K\Omega$ ) from RST to $V_{SS}$ permits power-on reset when a capacitor ( $\approx 10\mu f$ ) is also connected from this pin to $V_{CC}$ .             |
| <b>PORT 1</b><br>Port 1 is an 8-bit quasi-bidirectional I/O port. It is used for the low-order address byte during program verification. Port 1 can sink/source four LS TTL loads.  | <b>ALE/PROG</b><br>Provides Address Latch Enable output used for latching the address into external memory during normal operation. It is activated every six oscillator periods except during an external data memory access. It also receives the program pulse input for programming the EPROM version.          |
| In non-loop mode two of the I/O lines serve alternate functions:<br>—RTS (P1.6). Request-to-Send output. A low indicates that the RUPI-44 is ready to transmit.<br>—CTS (P1.7) Clear-to-Send input. A low indicates that a receiving station is ready to receive.   | <b>PSEN</b><br>The Program Store Enable output is a control signal that enables the external Program Memory to the bus during external fetch operations. It is activated every six oscillator periods, except during external data memory accesses. Remains high during internal program execution.                 |
| <b>PORT 2</b><br>Port 2 is an 8-bit quasi-bidirectional I/O port. It also emits the high-order address byte when accessing external memory. It is used for the high-order address and the control signals during program verification. Port 2 can sink/source four LS TTL loads.  | <b>EA/VPP</b><br>When held at a TTL high level, the RUPI-44 executes instructions from the internal ROM when the PC is less than 4096. When held at a TTL low level, the RUPI-44 fetches all instructions from external Program Memory. The pin also receives the 21V EPROM programming supply voltage on the 8744. |
| <b>PORT 3</b><br>Port 3 is an 8-bit quasi-bidirectional I/O port. It also contains the interrupt, timer, serial port and RD and WR pins that are used by various options. The output latch corresponding to a secondary function must be programmed to a one (1) for that function to operate. Port 3 can sink/source four LS TTL loads.  | <b>XTAL 1</b><br>Input to the oscillator's high gain amplifier. Required when a crystal is used. Connect to VSS when external source is used on XTAL 2.   |
| In addition to I/O, some of the pins also serve alternate functions as follows:<br>—I/O Rx/D (P3.0). In point-to-point or multipoint configurations, this pin controls the direction of pin P3.1. Serves as Receive Data input in loop and diagnostic modes.<br>—DATA Tx/D (P3.1) In point-to-point or multipoint configurations, this pin functions as data input/output. In loop mode, it serves as transmit pin. A '0' written to this pin enables diagnostic mode.<br>—INT0 (P3.2). Interrupt 0 input or gate control input for counter 0.<br>—INT1 (P3.3). Interrupt 1 input or gate control input for counter 1.<br>—TO (P3.4). Input to counter 0. | <b>XTAL 2</b><br>Output from the oscillator's amplifier. Input to the internal timing circuitry. A crystal or external source can be used.  |



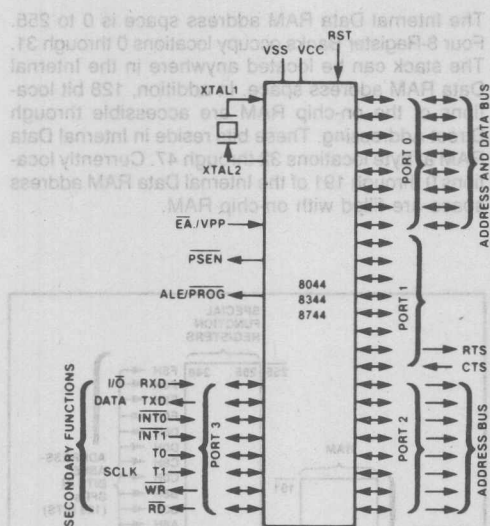


Figure 2.  
Logic Symbol

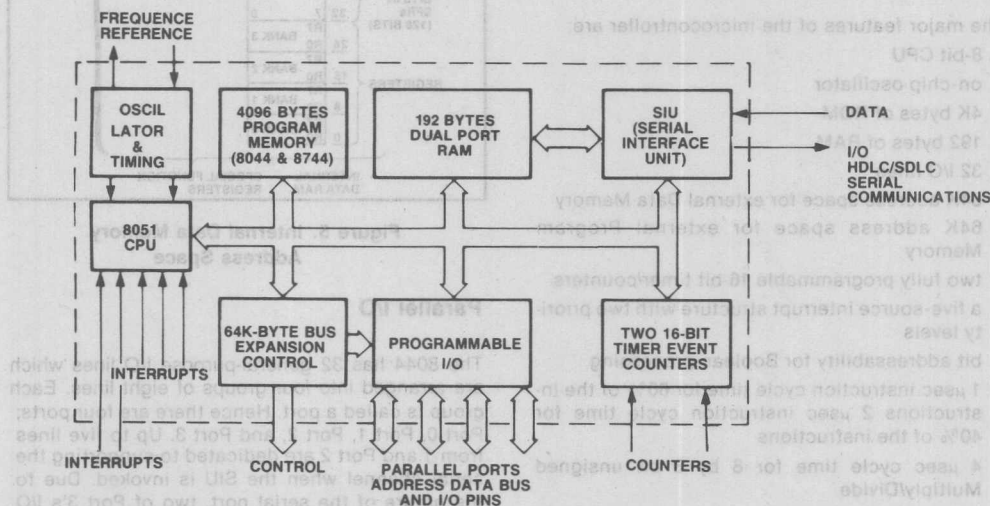


Figure 4.  
Block Diagram



## Functional Description

### General

The 8044 integrates the powerful 8051 microcontroller with an intelligent Serial Communication Controller to provide a single-chip solution which will efficiently implement a distributed processing or distributed control system. The microcontroller is a self-sufficient unit containing ROM, RAM, ALU, and its own peripherals. The 8044's architecture and instruction set are identical to the 8051's. The 8044 replaces the 8051's serial interface with an intelligent SDLC/HDLC Serial Interface Unit (SIU). 64 more bytes of RAM have been added to the 8051 RAM array. The SIU can communicate at bit rates up to 2.4 M bps. The SIU works concurrently with the Microcontroller so that there is no throughput loss in either unit. Since the SIU possesses its own intelligence, the CPU is off-loaded from many of the communications tasks, thus dedicating more of its computing power to controlling local peripherals or some external process.

### The Microcontroller

The microcontroller is a stand-alone high-performance single-chip computer intended for use in sophisticated real-time application such as instrumentation, industrial control, and intelligent computer peripherals.

The major features of the microcontroller are:

- 8-bit CPU
- on-chip oscillator
- 4K bytes of ROM
- 192 bytes of RAM
- 32 I/O lines
- 64K address space for external Data Memory
- 64K address space for external Program Memory
- two fully programmable 16-bit timer/counters
- a five-source interrupt structure with two priority levels
- bit addressability for Boolean processing
- 1  $\mu$ sec instruction cycle time for 60% of the instructions
- 2  $\mu$ sec instruction cycle time for 40% of the instructions
- 4  $\mu$ sec cycle time for 8 by 8 bit unsigned Multiply/Divide

### Internal Data Memory

Functionally the Internal Data Memory is the most flexible of the address spaces. The Internal Data Memory space is subdivided into a 256-byte Internal Data RAM address space and a 128-byte Special Function Register address space as shown in Figure 5.

The Internal Data RAM address space is 0 to 255. Four 8-Register Banks occupy locations 0 through 31. The stack can be located anywhere in the Internal Data RAM address space. In addition, 128 bit locations of the on-chip RAM are accessible through Direct Addressing. These bits reside in Internal Data RAM at byte locations 32 through 47. Currently locations 0 through 191 of the Internal Data RAM address space are filled with on-chip RAM.

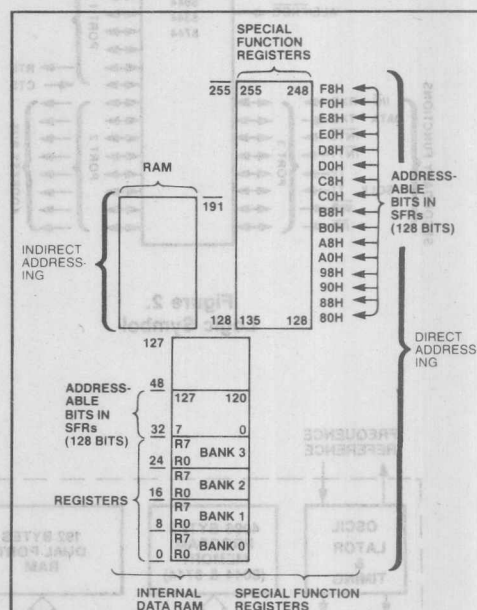


Figure 5. Internal Data Memory Address Space

### Parallel I/O

The 8044 has 32 general-purpose I/O lines which are arranged into four groups of eight lines. Each group is called a port. Hence there are four ports; Port 0, Port 1, Port 2, and Port 3. Up to five lines from 1 and Port 2 are dedicated to supporting the serial channel when the SIU is invoked. Due to the nature of the serial port, two of Port 3's I/O lines (P3.0 and P3.1) do not have latched outputs. This is true whether or not the serial channel is used.

Port 0 and Port 2 also have an alternate dedicated function. When placed in the external access mode, Port 0 and Port 2 become the means by

Table 1. MCS®-51 Instruction Set Description

| ARITHMETIC OPERATIONS |              |  |          | LOGICAL OPERATIONS (CONTINUED) |               |   |          |
|-----------------------|--------------|--|----------|--------------------------------|---------------|---|----------|
| Mnemonic              |              | Description                                | Byte Cyc | Mnemonic                       | Destination   |   | Byte Cyc |
| ADD                   | A,Rn         | Add register to Accumulator                | 1 1      | ORL                            | A,@Ri         | OR indirect RAM to Accumulator          | 1 1      |
| ADD                   | A,direct     | Add direct byte to Accumulator             | 2 1      | ORL                            | A,#data       | OR immediate data to Accumulator        | 2 1      |
| ADD                   | A,@Ri        | Add indirect RAM to Accumulator            | 1 1      | ORL                            | direct,A      | OR Accumulator to direct byte           | 2 1      |
| ADD                   | A,#data      | Add immediate data to Accumulator          | 2 1      | ORL                            | direct,#data  | OR immediate data to direct byte        | 3 2      |
| ADDC                  | A,Rn         | Add register to Accumulator with Carry     | 1 1      | XRL                            | A,Rn          | Exclusive-OR register to Accumulator    | 1 1      |
| ADDC                  | A,direct     | Add direct byte to A with Carry flag       | 2 1      | XRL                            | A,direct      | Exclusive-OR direct byte to Accumulator | 2 1      |
| ADDC                  | A,@Ri        | Add indirect RAM to A with Carry flag      | 1 1      | XRL                            | A,@Ri         | Exclusive-OR indirect RAM to A          | 1 1      |
| ADDC                  | A,#data      | Add immediate data to A with Carry flag    | 2 1      | XRL                            | A,#data       | Exclusive-OR immediate data to A        | 2 1      |
| SUBB                  | A,Rn         | Subtract register from A with Borrow       | 1 1      | XRL                            | direct,A      | Exclusive-OR Accumulator to direct byte | 2 1      |
| SUBB                  | A,direct     | Subtract direct byte from A with Borrow    | 2 1      | XRL                            | direct,#data  | Exclusive-OR immediate data to direct   | 3 2      |
| SUBB                  | A,@Ri        | Subtract indirect RAM from A with Borrow   | 1 1      | CLR                            | A             | Clear Accumulator                       | 1 1      |
| SUBB                  | A,#data      | Subtract immediate data from A with Borrow | 2 1      | CPL                            | A             | Complement Accumulator                  | 1 1      |
| INC                   | A            | Increment Accumulator                      | 1 1      | RL                             | A             | Rotate Accumulator Left                 | 1 1      |
| INC                   | Rn           | Increment register                         | 1 1      | RLC                            | A             | Rotate A Left through the Carry flag    | 1 1      |
| INC                   | direct       | Increment direct byte                      | 2 1      | RR                             | A             | Rotate Accumulator Right                | 1 1      |
| INC                   | @Ri          | Increment indirect RAM                     | 1 1      | RRC                            | A             | Rotate A Right through Carry flag       | 1 1      |
| INC                   | DPTR         | Increment Data Pointer                     | 1 2      | SWAP                           | A             | Swap nibbles within the Accumulator     | 1 1      |
| DEC                   | A            | Decrement Accumulator                      | 1 1      |                                |               |   |          |
| DEC                   | Rn           | Decrement register                         | 1 1      |                                |               |   |          |
| DEC                   | direct       | Decrement direct byte                      | 2 1      |                                |               |   |          |
| DEC                   | @Ri          | Decrement indirect RAM                     | 1 1      |                                |               |   |          |
| MUL                   | AB           | Multiply A & B                             | 1 4      |                                |               |   |          |
| DIV                   | AB           | Divide A by B                              | 1 4      |                                |               |   |          |
| DA                    | A            | Decimal Adjust Accumulator                 | 1 1      |                                |               |   |          |
| LOGICAL OPERATIONS    |              |  |          | DATA TRANSFER                  |               |   |          |
| Mnemonic              |              | Destination                                | Byte Cyc | Mnemonic                       | Description   |   | Byte Cyc |
| ANL                   | A,Rn         | AND register to Accumulator                | 1 1      | MOV                            | A,Rn          | Move register to Accumulator            | 1 1      |
| ANL                   | A,direct     | AND direct byte to Accumulator             | 2 1      | MOV                            | A,direct      | Move direct byte to Accumulator         | 2 1      |
| ANL                   | A,@Ri        | AND indirect RAM to Accumulator            | 1 1      | MOV                            | A,@Ri         | Move indirect RAM to Accumulator        | 1 1      |
| ANL                   | A,#data      | AND immediate data to Accumulator          | 2 1      | MOV                            | A,#data       | Move immediate data to Accumulator      | 2 1      |
| ANL                   | direct,A     | AND Accumulator to direct byte             | 2 1      | MOV                            | Rn,A          | Move Accumulator to register            | 1 1      |
| ANL                   | direct,#data | AND immediate data to direct byte          | 3 2      | MOV                            | Rn,direct     | Move direct byte to register            | 2 2      |
| ORL                   | A,Rn         | OR register to Accumulator                 | 1 1      | MOV                            | Rn,#data      | Move immediate data to register         | 2 1      |
| ORL                   | A,direct     | OR direct byte to Accumulator              | 2 1      | MOV                            | direct,A      | Move Accumulator to direct byte         | 2 1      |
|                       |              |  |          | MOV                            | direct,Rn     | Move register to direct byte            | 2 2      |
|                       |              |  |          | MOV                            | direct,direct | Move direct byte to direct              | 3 2      |
|                       |              |  |          | MOV                            | direct,@Ri    | Move indirect RAM to direct byte        | 2 2      |

Table 1. (Cont.)

| DATA TRANSFER (CONTINUED)                          |              |   |      | PROGRAM AND MACHINE CONTROL |   |   |  |      |     |
|--|--------------|---|------|-----------------------------|---|---|--|------|-----|
| Mnemonic   |              | Description                               | Byte | Cyc                         | Mnemonic                                  |   | Description                              | Byte | Cyc |
| MOV  | direct,#data | Move immediate data to direct byte        | 3    | 2                           | ACALL                                     | addr11  | Absolute Subroutine Call                 | 2    | 2   |
| MOV  | @Ri,A        | Move Accumulator to indirect RAM          | 1    | 1                           | LCALL                                     | addr16  | Long Subroutine Call                     | 3    | 2   |
| MOV  | @Ri,direct   | Move direct byte to indirect RAM          | 2    | 2                           | RET                                       |   | Return from subroutine                   | 1    | 2   |
| MOV  | @Ri,#data    | Move immediate data to indirect RAM       | 2    | 1                           | RETI                                      |   | Return from interrupt                    | 1    | 2   |
| MOV  | DPTR,#data16 | Load Data Pointer with a 16-bit constant  | 3    | 2                           | AJMP                                      | addr11  | Absolute Jump                            | 2    | 2   |
| MOVC   | A,@A+DPTR    | Move Code byte relative to DPTR to A      | 1    | 2                           | LJMP                                      | addr16  | Long Jump                                | 3    | 2   |
| MOVC   | A,@A+PC      | Move Code byte relative to PC to A        | 1    | 2                           | SJMP                                      | rel   | Short Jump (relative addr)               | 2    | 2   |
| MOVB   | A,@Ri        | Move External RAM (8-bit addr) to A       | 1    | 2                           | JMP                                       | @A+DPTR   | Jump indirect relative to the DPTR       | 1    | 2   |
| MOVB   | A,@DPTR      | Move External RAM (16-bit addr) to A      | 1    | 2                           | JZ  | rel   | Jump if Accumulator is Zero              | 2    | 2   |
| MOVX   | @Ri,A        | Move A to External RAM (8-bit addr)       | 1    | 2                           | JNZ                                       | rel   | Jump if Accumulator is Not Zero          | 2    | 2   |
| MOVX   | @DPTR,A      | Move A to External RAM (16-bit addr)      | 1    | 2                           | JC  | rel   | Jump if Carry flag is set                | 2    | 2   |
| PUSH   | direct       | Push direct byte onto stack               | 2    | 2                           | JNC                                       | rel   | Jump if No Carry flag                    | 2    | 2   |
| POP  | direct       | Pop direct byte from stack                | 2    | 2                           | JB  | bit,rel   | Jump if direct Bit set                   | 3    | 2   |
| XCH  | A,Rn         | Exchange register with Accumulator        | 1    | 1                           | JNB                                       | bit,rel   | Jump if direct Bit Not set               | 3    | 2   |
| XCH  | A,direct     | Exchange direct byte with Accumulator     | 2    | 1                           | JBC                                       | bit,rel   | Jump if direct Bit is set & Clear bit    | 3    | 2   |
| XCH  | A,@Ri        | Exchange indirect RAM with A              | 1    | 1                           | CJNE                                      | A,direct,rel  | Compare direct to A & Jump if Not Equal  | 3    | 2   |
| XCHD   | A,@Ri        | Exchange low-order Digit ind RAM w A      | 1    | 1                           | CJNE                                      | A,#data,rel   | Comp, immed, to A & Jump if Not Equal    | 3    | 2   |
| BOOLEAN VARIABLE MANIPULATION                      |              |   |      |                             | CJNE                                      | Rn,#data,rel  | Comp, immed, to reg & Jump if Not Equal  | 3    | 2   |
| Mnemonic   |              | Description                               | Byte | Cyc                         | CJNE                                      | @Ri,#data,rel   | Comp, immed, to ind, & Jump if Not Equal | 3    | 2   |
| CLR  | C            | Clear Carry flag                          | 1    | 1                           | DJNZ                                      | Rn,rel  | Decrement register & Jump if Not Zero    | 2    | 2   |
| CLR  | bit          | Clear direct bit                          | 2    | 1                           | DJNZ                                      | direct,rel  | Decrement direct & Jump if Not Zero      | 3    | 2   |
| SETB   | C            | Set Carry flag                            | 1    | 1                           | NOP                                       |   | No operation                             | 1    | 1   |
| SETB   | bit          | Set direct Bit                            | 2    | 1                           | <b>Notes on data addressing modes:</b>    |   |  |      |     |
| CPL  | C            | Complement Carry flag                     | 1    | 1                           | Rn  | —Working register R0–R7   |  |      |     |
| CPL  | bit          | Complement direct bit                     | 2    | 1                           | direct                                    | —128 internal RAM locations, any I/O port, control or status register   |  |      |     |
| ANL  | C,bit        | AND direct bit to Carry flag              | 2    | 2                           | @Ri                                       | —Indirect internal RAM location addressed by register R0 or R1  |  |      |     |
| ANL  | C,/bit       | AND complement of direct bit to Carry     | 2    | 2                           | #data                                     | —8-bit constant included in instruction   |  |      |     |
| ORL  | C,/bit       | OR complement of direct bit to Carry flag | 2    | 2                           | #data16                                   | —16-bit constant included as bytes 2 & 3 of instruction   |  |      |     |
| ORL  | C,bit        | OR direct bit to Carry flag               | 2    | 2                           | bit                                       | —128 software flags, any I/O pin, control or status bit   |  |      |     |
| MOV  | C,/bit       | Move direct bit to Carry flag             | 2    | 1                           | <b>Notes on program addressing modes:</b> |   |  |      |     |
| MOV  | bit,C        | Move Carry flag to direct bit             | 2    | 2                           | addr16                                    | —Destination address for LCALL & LJMP may be anywhere within the 64-K program memory address space  |  |      |     |
|  |              |   |      |                             | Addr11                                    | —Destination address for ACALL & AJMP will be within the same 2-K page of program memory as the first byte of the following instruction   |  |      |     |
|  |              |   |      |                             | rel                                       | —SJMP and all conditional jumps include an 8-bit offset byte, Range is +127–128 bytes relative to first byte of the following instruction |  |      |     |
| All mnemonics copyrighted © Intel Corporation 1979 |              |   |      |                             |   |   |  |      |     |

which the 8044 communicates with external program memory. Port 0 and Port 2 are also the means by which the 8044 communicates with external data memory. Peripherals can be memory mapped into the address space and controlled by the 8044.

### Timer/Counters

The 8044 contains two 16-bit counters which can be used for measuring time intervals, measuring pulse widths, counting events, generating precise periodic interrupt requests, and clocking the serial communications. Internally the Timers are clocked at 1/12 of the crystal frequency, which is the instruction cycle time. Externally the counters can run up to 500 KHz.

### Interrupt System

External events and the real-time driven on-chip peripherals require service by the CPU asynchronous to the execution of any particular section of code. To tie the asynchronous activities of these functions to normal program execution, a sophisticated multiple-source, two priority level, nested interrupt system is provided. Interrupt response latency ranges from 3  $\mu$ sec to 7  $\mu$ sec when using a 12 MHz clock.

All five interrupt sources can be mapped into one of the two priority levels. Each interrupt source can be enabled or disabled individually or the entire interrupt system can be enabled or disabled. The five interrupt sources are: Serial Interface Unit, Timer 1, Timer 2, and two external interrupts. The external interrupts can be either level or edge triggered.

### Serial Interface Unit (SIU)

The Serial Interface Unit is used for HDLC/SDLC communications. It handles Zero Bit Insertion/Deletion, Flags, automatic address recognition, and a 16-bit cyclic redundancy check. In addition it implements in hardware a subset of the SDLC protocol such that it responds to many SDLC frames without CPU intervention. In certain applications it is advantageous to have the CPU control the reception or transmission of every single frame. For this reason the SIU has two modes of operation: "AUTO" and "FLEXIBLE" (or "NON-AUTO"). It is in the AUTO mode that the SIU responds to SDLC frames without CPU intervention; whereas, in the FLEXIBLE mode the reception or transmission of every single frame will be under CPU control.

There are three control registers and eight parameter registers that are used to operate the serial interface. These registers are shown in Figure 5 and Figure 6. The control registers set the modes of

operation and provide status information. The eight parameter registers buffer the station address, receive and transmit control bytes, and point to the on-chip transmit and receive buffers.

Data to be received or transmitted by the SIU must be buffered anywhere within the 192 bytes of on-chip RAM. Transmit and receive buffers are not allowed to "wrap around" in RAM; a "buffer end" is generated after address 191 is reached.

With the addition of only a few bytes of code, the 8044's frame size is not limited to the size of its internal RAM (192 bytes), but rather by the size of external buffer with no degradation of the RUP's features (e.g. NRZI, zero bit insertion/deletion, address recognition, cyclic redundancy check). There is a special function register called SIUST whose contents dictates the operation of the SIU. At low data rates, one section of the SIU (the Byte Processor) performs no function during known intervals. For a given data rate, these intervals (stand-by mode) are fixed. The above characteristics make it possible to program the CPU to move data to/from external RAM and to force the SIU to perform some desired hardware tasks while transmission or reception is taking place. With these modifications, external RAM can be utilized as a transmit and receive buffer instead of the internal RAM.

### AUTO Mode

In the AUTO mode the SIU implements in hardware a subset of the SDLC protocol such that it responds to many SDLC frames without CPU intervention. All AUTO mode responses to the primary station will conform to IBM's SDLC definition. The advantages of the AUTO mode are that less software is required to implement a secondary station, and the hardware generated response to polls is much faster than doing it in software. However, the Auto mode can not be used at a primary station.

To transmit in the AUTO mode the CPU must load the Transmit Information Buffer, Transmit Buffer Start register, Transmit Buffer Length register, and set the Transmit Buffer Full bit. The SIU automatically responds to a poll by transmitting an information frame with the P/F bit in the control field set. When the SIU receives a positive acknowledgement from the primary station, it automatically increments the Ns field in the NSNR register and interrupts the CPU. A negative acknowledgement would cause the SIU to retransmit the frame.

To receive in the AUTO mode, the CPU loads the Receive Buffer Start register, the Receive Buffer Length register, clears the Receive Buffer Protect bit, and sets the Receive Buffer Empty bit. If the







SIU is polled in this state, and the TBF bit indicates that the Transmit Buffer is empty, an automatic RR response will be generated. When a valid information frame is received the SIU will automatically increment Nr in the NSNR register and interrupt the CPU.

While in the AUTO mode the SIU can recognize and respond to the following commands without CPU intervention: I (Information), RR (Receive Ready), RNR (Receive Not Ready), REJ (Reject), and UP (Unnumbered Poll). The SIU can generate the following responses without CPU intervention: I (Information), RR (Receive Ready), and RNR (Receive Not Ready).

When the Receive Buffer Empty bit (RBE) indicates that the Receive Buffer is empty, the receiver is enabled, and when the RBE bit indicates that the Receive Buffer is full, the receiver is disabled. Assuming that the Receive Buffer is empty, the SIU will respond to a poll with an I frame if the Transmit Buffer is full. If the Transmit Buffer is empty, the SIU will respond to a poll with a RR command if the Receive Buffer Protect bit (RBP) is cleared, or an RNR command if RBP is set.

#### FLEXIBLE (or NON-AUTO) Mode

In the FLEXIBLE mode all communications are under control of the CPU. It is the CPU's task to encode and decode control fields, manage acknowledgements, and adhere to the requirements of the HDLC/SDLC protocols. The 8044 can be used as a primary or a secondary station in this mode.

To receive a frame in the FLEXIBLE mode, the CPU must load the Receive Buffer Start register, the Receive Buffer Length register, clear the Receive Buffer Protect bit, and set the Receive Buffer Empty bit. If a valid opening flag is received and the address field matches the byte in the Station Address register or the address field contains a broadcast address, the 8044 loads the control field in the receive control byte register, and loads the I field in the receive buffer. If there is no CRC error, the SIU interrupts the CPU, indicating a frame has just been received. If there is a CRC error, no interrupt occurs. The Receive Field Length register provides the number of bytes that were received in the information field.

To transmit a frame, the CPU must load the transmit information buffer, the Transmit Buffer Start register, the Transmit Buffer Length register, the Transmit Control Byte, and set the TBF and the RFS bit. The SIU, unsolicited by an HDLC/SDLC frame, will transmit the entire information frame, and interrupt the CPU, indicating the completion of transmission. For supervisory frames or unnumbered frames, the

transmit buffer length would be 0.

#### CRC

The FCS register is initially set to all 1's prior to calculating the FCS field. The SIU will not interrupt the CPU if a CRC error occurs (in both AUTO and FLEXIBLE modes). The CRC error is cleared upon receiving of an opening flag.

#### Frame Format Options

In addition to the standard SDLC frame format, the 8044 will support the frames displayed in Figure 7. The standard SDLC frame is shown at the top of this figure. For the remaining frames the information field will incorporate the control or address bytes and the frame check sequences; therefore these fields will be stored in the Transmit and Receive buffers. For example, in the non-buffered mode the third byte is treated as the beginning of the information field. In the non-addressed mode, the information field begins after the opening flag. The mode bits to set the frame format options are found in the Serial Mode register and the Status register.

#### EXTENDED ADDRESSING

To realize an extended control field or an extended address field using the HDLC protocol, the FLEXIBLE mode must be used. For an extended control field, the SIU is programmed to be in the non-buffered mode. The extended control field will be the first and second bytes in the Receive and Transmit Buffers. For extended addressing the SIU is placed in the non-addressed mode. In this mode the CPU must implement the address recognition for received frames. The addressing field will be the initial bytes in the Transmit and Receive buffers followed by the control field.

The SIU can transmit and receive only frames which are multiples of 8 bits. For frames received with other than 8-bit multiples, a CRC error will cause the SIU to reject the frame.

#### SDLC Loop Networks

The SIU can be used in an SDLC loop as a secondary or primary station. When the SIU is placed in the Loop mode it receives the data on pin 10 and transmits the data one bit time delayed on pin 11. It can also recognize the Go ahead signal and change it into a flag when it is ready to transmit. As a secondary station the SIU can be used in the AUTO or FLEXIBLE modes. As a primary station the FLEXIBLE mode is used; however, additional hardware is required for generating the Go Ahead bit pattern. In the Loop mode the maximum data rate is 1 Mbps clocked or 375 Kbps self-clocked.

| FRAME OPTION  | NFCS | NB | AM | FRAME FORMAT               |
|---|------|----|----|----------------------------|
| Standard SDLC<br>NON-AUTO Mode  | 0    | 0  | 0  |                            |
| Standard SDLC<br>AUTO Mode  | 0    | 0  | 1  |                            |
| Non-Buffered Mode<br>NON-AUTO Mode  | 0    | 1  | 1  |                            |
| Non-Addressed Mode<br>NON-AUTO Mode   | 0    | 1  | 0  |                            |
| No FCS Field<br>NON-AUTO Mode   | 1    | 0  | 0  |                            |
| No FCS Field<br>AUTO Mode   | 1    | 0  | 1  |                            |
| No FCS Field<br>Non-Buffered Mode<br>NON-AUTO Mode  | 1    | 1  | 1  |                            |
| No FCS Field<br>Non-Addressed Mode<br>NON-AUTO Mode   | 1    | 1  | 0  |                            |
| Mode Bits:  |      |    |    |                            |
| AM — "AUTO" Mode/Addressed Mode   |      |    |    |                            |
| NB — Non-Buffered Mode  |      |    |    |                            |
| NFCS — No FCS Field Mode  |      |    |    |                            |
| Key to Abbreviations:   |      |    |    |                            |
| F = Flag (01111110)   |      |    |    | I = Information Field      |
| A = Address Field   |      |    |    | FCS = Frame Check Sequence |
| C = Control Field   |      |    |    |                            |
| Note 1: The AM bit function is controlled by the NB bit. When NB = 0, AM becomes AUTO mode select, when NB = 1, AM becomes Address mode select. |      |    |    |                            |

Figure 7. Frame Format Options

### SDLC Multidrop Networks

The SIU can be used in a SDLC non-loop configuration as a secondary or primary station. When the SIU is placed in the non-loop mode, data is received and transmitted on pin 11, and pin 10 drives a tri-state buffer. In non-loop mode, modem interface pins, RTS and CTS, become available.

### Data Clocking Options

The 8044's serial port can operate in an externally clocked or self clocked system. A clocked system provides to the 8044 a clock synchronization to the data. A self-clocked system uses the 8044's on-chip Digital Phase Locked Loop (DPLL) to recover the clock from the data, and clock this data into the Serial Receive Shift Register.

In this mode, a clock synchronized with the data is externally fed into the 8044. This clock may be generated from an External Phase Locked Loop, or possibly supplied along with the data. The 8044 can transmit and receive data in this mode at rates up to 2.4 Mbps.

This self clocked mode allows data transfer without

a common system data clock. An on-chip Digital Phase Locked Loop is employed to recover the data clock which is encoded in the data stream. The DPLL will converge to the nominal bit center within eight bit transitions, worst case. The DPLL requires a reference clock of either 16 times (16x) or 32 times (32x) the data rate. This reference clock may be externally applied or internally generated. When internally generated either the 8044's internal logic clock (crystal frequency divided by two) or the timer 1 overflow is used as the reference clock. Using the internal timer 1 clock the data rates can vary from 244 to 62.5 Kbps. Using the internal logic clock at a 16x sampling rate, receive data can either be 187.5 Kbps, or 375 Kbps. When the reference clock for the DPLL is externally applied the data rates can vary from 0 to 375 Kbps at a 16x sampling rate.

To aid in a Phase Locked Loop capture, the SIU has a NRZI (Non Return to Zero Inverted) data encoding and decoding option. Additionally the SIU has a pre-frame sync option that transmits two bytes of alternating 1's and 0's to ensure that the receive station DPLL will be synchronized with the data by the time it receives the opening flag.

## Control and Status Registers

There are three SIU Control and Status Registers:

Serial Mode Register (SMD)

Status/Command Register (STS)

Send/Receive Count Register (NSNR)

The SMD, STS, and NSNR registers are all cleared by system reset. This assures that the SIU will power up in an idle state (neither receiving nor transmitting).

These registers and their bit assignments are described below.

### SMD: Serial Mode Register (byte-addressable)

|        |      |      |      |      |     |    |      |
|--------|------|------|------|------|-----|----|------|
| Bit: 7 | 6    | 5    | 4    | 3    | 2   | 1  | 0    |
| SCM2   | SCM1 | SCM0 | NRZI | LOOP | PFS | NB | NFCS |

The Serial Mode Register (Address C9H) selects the operational modes of the SIU. The 8044 CPU can both read and write SMD. The SIU can read SMD but cannot write to it. To prevent conflict between CPU and SIU access to SMD, the CPU should write SMD only when the Request To Send (RTS) and Receive Buffer Empty (RBE) bits (in the STS register) are both false (0). Normally, SMD is accessed only during initialization.

The individual bits of the Serial Mode Register are as follows:

| Bit # | Name | Description   |
|-------|------|---|
| SMD.0 | NFCS | No FCS field in the SDLC frame.   |
| SMD.1 | NB   | Non-Buffered mode. No control field in the SDLC frame.  |
| SMD.2 | PFS  | Pre-Frame Sync mode. In this mode, the 8044 transmits two bytes before the first flag of a frame, for DPLL synchronization. If NRZI is enabled, 00H is sent; otherwise, 55H is sent. In either case, 16 pre-frame transitions are guaranteed. |
| SMD.3 | LOOP | Loop configuration.   |
| SMD.4 | NRZI | NRZI coding option.<br>If bit = 1, NRZI coding is used. If bit = 0, then it is straight binary (NRZ).   |
| SMD.5 | SCM0 | Select Clock Mode — Bit 0   |
| SMD.6 | SCM1 | Select Clock Mode — Bit 1   |
| SMD.7 | SCM2 | Select Clock Mode — Bit 2   |

The SCM bits decode as follows:

| SCM   | Clock Mode                   | Data Rate (Bits/sec)* |
|-------|------------------------------|-----------------------|
| 2 1 0 |                              |                       |
| 0 0 0 | Externally clocked           | 0-2.4M**              |
| 0 0 1 | Reserved                     |                       |
| 0 1 0 | Self clocked, timer overflow | 244-62.5K             |

|       |                              |                       |
|-------|------------------------------|-----------------------|
| 0 1 1 | Reserved                     |                       |
| 1 0 0 | Self clocked, external 16x   | 0-375K                |
| SCM   |                              | Data Rate (Bits/sec)* |
| 2 1 0 | Clock Mode                   |                       |
| 1 0 1 | Self clocked, external 32x   | 0-187.5K              |
| 1 1 0 | Self clocked, internal fixed | 375K                  |
| 1 1 1 | Self clocked, internal fixed | 187.5k                |

\*Based on a 12 Mhz crystal frequency

\*\*0-1M bps in loop configuration

### STS: Status/Command Register (bit-addressable)

|        |     |     |    |     |     |    |     |
|--------|-----|-----|----|-----|-----|----|-----|
| Bit: 7 | 6   | 5   | 4  | 3   | 2   | 1  | 0   |
| TBF    | RBE | RTS | SI | BOV | OPB | AM | RBP |

The Status/Command Register (Address C8H) provides operational control of the SIU by the 8044 CPU, and enables the SIU to post status information for the CPU's access. The SIU can read STS, and can alter certain bits, as indicated below. The CPU can both read and write STS asynchronously. However, 2-cycle instructions that access STS during both cycles ('JBC/B, REL' and 'MOV /B.C.') should not be used, since the SIU may write to STS between the two CPU accesses.

The individual bits of the Status/Command Register are as follows:

| Bit # | Name | Description   |
|-------|------|---|
| STS.0 | RBP  | Receive Buffer Protect. Inhibits writing of data into the receive buffer. In AUTO mode, RBP forces an RNR response instead of an RR.  |
| STS.1 | AM   | AUTO Mode/Addressed Mode. Selects AUTO mode where AUTO mode is allowed. If NB is true, (=1), the AM bit selects the addressed mode. AM may be cleared by the SIU.                                     |
| STS.2 | OPB  | Optional Poll Bit. Determines whether the SIU will generate an AUTO response to an optional poll (UP with P=0). OPB may be set or cleared by the SIU.   |
| STS.3 | BOV  | Receive Buffer Overrun. BOV may be set or cleared by the SIU.   |
| STS.4 | SI   | SIU Interrupt. This is one of the five interrupt sources to the CPU. The vector location = 23H. SI may be set by the SIU. It should be cleared by the CPU before returning from an interrupt routine. |
| STS.5 | RTS  | Request To Send. Indicates that the 8044 is ready to transmit or is transmitting. RTS may be read or written by the CPU. RTS may be read by the SIU, and in AUTO mode may be written by the SIU.      |

- STS.6 RBE Receive Buffer Empty. RBE can be thought of as Receive Enable. RBE is set to one by the CPU when it is ready to receive a frame, or has just read the buffer, and to zero by the SIU when a frame has been received.
- STS.7 TBF Transmit Buffer Full. Written by the CPU to indicate that it has filled the transmit buffer. TBF may be cleared by the SIU.

**NSNR: Send/Receive Count Register (bit-addressable)**

|      |     |     |     |     |     |     |     |     |
|------|-----|-----|-----|-----|-----|-----|-----|-----|
| Bit: | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |
|      | NS2 | NS1 | NS0 | SES | NR2 | NR1 | NR0 | SER |

The Send/Receive Count Register (Address D8H) contains the transmit and receive sequence numbers, plus tally error indications. The SIU can both read and write NSNR. The 8044 CPU can both read and write NSNR asynchronously. However, 2-cycle instructions that access NSNR during both cycles ('JBC /B, REL', and 'MOV /B,C') should not be used, since the SIU may write to NSNR between the two 8044 CPU accesses.

The individual bits of the Send/Receive Count Register are as follows:

| Bit #  | Name | Description  |
|--------|------|--|
| NSNR.0 | SER  | Receive Sequence Error:<br>NS (P) $\neq$ NR (S)                              |
| NSNR.1 | NR0  | Receive Sequence Counter—Bit 0   |
| NSNR.2 | NR1  | Receive Sequence Counter—Bit 1   |
| NSNR.3 | NR2  | Receive Sequence Counter—Bit 2   |
| NSNR.4 | SES  | Send Sequence Error:<br>NR (P) $\neq$ NS (S) and<br>NR (P) $\neq$ NS (S) + 1 |
| NSNR.5 | NS0  | Send Sequence Counter — Bit 0  |
| NSNR.6 | NS1  | Send Sequence Counter — Bit 1  |
| NSNR.7 | NS2  | Send Sequence Counter — Bit 2  |

**Parameter Registers**

There are eight parameter registers that are used in connection with SIU operation. All eight registers may be read or written by the 8044 CPU. RFL and RCB are normally loaded by the SIU.

The eight parameter registers are as follows:

**STAD: Station Address Register (byte-addressable)**

The Station Address register (Address CEH) contains the station address. To prevent access conflict, the CPU should access STAD only when the SIU is idle (RTS=0

and RBE=0). Normally, STAD is accessed only during initialization.

**TBS: Transmit Buffer Start Address Register (byte-addressable)**

The Transmit Buffer Start address register (Address DCH) points to the location in on-chip RAM for the beginning of the I-field of the frame to be transmitted. The CPU should access TBS only when the SIU is not transmitting a frame (when TBF=0).

**TBL: Transmit Buffer Length Register (byte-addressable)**

The Transmit Buffer Length register (Address DBH) contains the length (in bytes) of the I-field to be transmitted. A blank I-field (TBL=0) is valid. The CPU should access TBL only when the SIU is not transmitting a frame (when TBF=0).

NOTE: The transmit and receive buffers are not allowed to "wrap around" in the on-chip RAM. A "buffer end" is automatically generated if address 191 (BFH) is reached.

**TCB: Transmit Control Byte Register (byte-addressable)**

The Transmit Control Byte register (Address DAH) contains the byte which is to be placed in the control field of the transmitted frame, during NON-AUTO mode transmission. The CPU should access TCB only when the SIU is not transmitting a frame (when TBF=0). The  $N_S$  and  $N_R$  counters are not used in the NON-AUTO mode.

**RBS: Receive Buffer Start Address Register (byte-addressable)**

The Receive Buffer Start address register (Address CCH) points to the location in on-chip RAM where the beginning of the I-field of the frame being received is to be stored. The CPU should write RBS only when the SIU is not receiving a frame (when RBE=0).

**RBL: Receive Buffer Length Register (byte-addressable)**

The Receive Buffer Length register (Address CBH) contains the length (in bytes) of the area in on-chip

RAM allocated for the received I-field. RBL=0 is valid. The CPU should write RBL only when RBE=0.

**RFL: Receive Field Length Register (byte-addressable)**

The Received Field Length register (Address CDH) contains the length (in bytes) of the received I-field that has just been loaded into on-chip RAM. RFL is loaded by the SIU. RFL=0 is valid. RFL should be accessed by the CPU only when RBE=0.



# RCB: Receive Control Byte Register (byte-addressable)

The Received Control Byte register (Address CAH) contains the control field of the frame that has just been received. RCB is loaded by the SIU. The CPU can only read RCB, and should only access RCB when RBE=0.

## ICE Support

The 8044 In-Circuit Emulator (ICE-44) allows the user to exercise the 8044 application system and monitor the execution of instructions in real time.

The emulator operates with Intel's Intellec<sup>®</sup> development system. The development system interfaces with the user's 8044 system through an in-cable buffer box. The cable terminates in a 8044 pin-compatible plug, which fits into the 8044 socket in the user's system. With

the emulator plug in place, the user can exercise his system in real time while collecting up to 255 instruction cycles of real-time data. In addition, he can single-step the program.

Static RAM is available (in the in-cable buffer box) to emulate the 8044 internal and external program memory and external data memory. The designer can display and alter the contents of the replacement memory in the buffer box, the internal data memory, and the internal 8044 registers, including the SFRs.

## SIUST: SIU State Counter (byte-addressable)

The SIU State Counter (Address D9H) reflects the state of the internal logic which is under SIU control. Therefore, care must be taken not to write into this register. This register provides a useful means for debugging 8044 receiver problem.

| Symbol | Parameter  | Units | Typical | Maximum | Minimum |
|--------|--|-------|---------|---------|---------|
| CIO    | Pin Capacitance  | pF    | 10      |         |         |
| ICC    | Power Supply Current   | mA    | 285     |         |         |
| IIR1   | Input Current to RST to Activate Reset                       | μA    | 800     |         |         |
| IIR    | Logical 1 Input Current to EA Pin of 8744H                   | μA    | -500    |         |         |
| ILI    | Input Leakage Current (Port 0)                               | μA    | ±100    |         |         |
| ILS    | Logical 0 Input Current (XTAL2)                              | mA    | -3.8    |         |         |
| IL1    | Logical 0 Input Current to EA Pin of 8744H only              | mA    | -15     |         |         |
| IL     | Logical 0 Input Current (Ports 1, 2, 3)                      | μA    | -500    |         |         |
| VOH1   | Output High Voltage (Port 0 in External Bus Mode, ALE, PSEN) | V     |         | 2.4     |         |
| VOH    | Output High Voltage (Ports 1, 2, 3)                          | V     |         | 2.4     |         |
| VOL1   | Output Low Voltage (Port 0, ALE, PSEN)                       | V     |         | 0.45    |         |
| VOL    | Output Low Voltage (Ports 1, 2, 3)                           | V     |         | 0.45    |         |

\*Notes: Capacitive loading on Ports 0 and 2 may cause spurious noise pulses to be superimposed on the VOLS of ALE and Ports 1 and 2. The noise is due to external bus capacitance discharging into the Port 0 and Port 2 pins when these pins make 1-to-0 transitions during bus operations. In the worst cases (capacitive loading > 100 pF), the noise pulse on the ALE line may exceed 0.5V. In such cases it may be desirable to qualify ALE with a Schmitt Trigger or use an address latch with Schmitt Trigger STROBE input.



# **ABSOLUTE MAXIMUM RATINGS\***

Ambient Temperature Under Bias .... 0°C to 70°C  
 Storage Temperature ..... -65°C to -150°C  
 Voltage on EA, VPP Pin to VSS .... -0.5V to -21.5V  
 Voltage on Any Other Pin to VSS .... -0.5V to -7V  
 Power Dissipation.....2W

*\*NOTICE: Stresses above those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

## **D.C. CHARACTERISTICS:** (T<sub>A</sub> = 0°C to 70°C, VCC = 5V = 10%, VSS = 0V)

| Symbol | Parameter  | Min  | Max       | Unit | Test Conditions                    |
|--------|--|------|-----------|------|------------------------------------|
| VIL    | Input Low Voltage (Except EA Pin of 8744H)                   | -0.5 | 0.8       | V    |                                    |
| VIL1   | Input Low Voltage to EA Pin of 8744H                         | 0    | 0.8       | V    |                                    |
| VIH    | Input High Voltage (Except XTAL2, RST)                       | 2.0  | VCC - 0.5 | V    |                                    |
| VIH1   | Input High Voltage to XTAL2, RST                             | 2.5  | VCC - 0.5 | V    | XTAL1 = VSS                        |
| VOL    | Output Low Voltage (Ports 1, 2, 3)*                          |      | 0.45      | V    | IOL = 1.6mA                        |
| VOL1   | Output Low Voltage (Port 0, ALE, PSEN)*                      |      |           |      |                                    |
|        | 8744H  |      | 0.60      | V    | IOL = 3.2 mA                       |
|        | 8044AH/8344AH  |      | 0.45      | V    | IOL = 2.4 mA                       |
| VOH    | Output High Voltage (Ports 1, 2, 3)                          | 2.4  |           | V    | IOH = -80 µA                       |
| VOH1   | Output High Voltage (Port 0 in External Bus Mode, ALE, PSEN) | 2.4  |           | V    | IOH = -400 µA                      |
| IIL    | Logical 0 Input Current (Ports 1, 2, 3)                      |      | -500      | µA   | Vin = 0.45 V                       |
| IIL1   | Logical 0 Input Current to EA Pin of 8744H only              |      | -15       | mA   |                                    |
| IIL2   | Logical 0 Input Current (XTAL2)                              |      | -3.6      | mA   | Vin = 0.45 V                       |
| ILI    | Input Leakage Current (Port 0)                               |      | ±100      | µA   | 0.45 < Vin < VCC                   |
|        |  |      | ±10       | µA   | 0.45 < Vin < VCC                   |
| IIH    | Logical 1 Input Current to EA Pin of 8744H                   |      | 500       | µA   |                                    |
| IIH1   | Input Current to RST to Activate Reset                       |      | 500       | µA   | Vin < (VCC - 1.5V)                 |
| ICC    | Power Supply Current:  |      | 285       | mA   | All Outputs Disconnected: EA = VCC |
|        |  |      | 170       | mA   |                                    |
| CIO    | Pin Capacitance  |      | 10        | pF   | Test Freq. = 1MHz                  |

**\*Notes:** Capacitive loading on Ports 0 and 2 may cause spurious noise pulses to be superimposed on the VOLs of ALE and Ports 1 and 3. The noise is due to external bus capacitance discharging into the Port 0 and Port 2 pins when these pins make 1-to-0 transitions during bus operations. In the worst cases (capacitive loading > 100 pF), the noise pulse on the ALE line may exceed 0.8V. In such cases it may be desirable to qualify ALE with a Schmitt Trigger, or use an address latch with a Schmitt Trigger STROBE input.

**A.C. CHARACTERISTICS:** ( $T_A = 0^\circ\text{C}$  to  $+70^\circ\text{C}$ ,  $V_{CC} = 5V \pm 10\%$ ,  $V_{SS} = 0V$ )

Load Capacitance for Port 0, ALE, and PSEN = 100 pF,

Load Capacitance for All Other Outputs = 80 pF)

**EXTERNAL PROGRAM MEMORY CHARACTERISTICS**

| Symbol             | Parameter  | 12MHz Osc |     | Variable Clock<br>1/TCLCL = 1.2MHz to 12MHz |            | Unit |
|--------------------|--|-----------|-----|---|------------|------|
|                    |  | Min       | Max | Min   | Max        |      |
| TLHLL              | ALE Pulse Width                                      | 127       |     | 2TCLCL-40                                   |            | ns   |
| TAVLL              | Address Valid to ALE Low                             | 43        |     | TCLCL-40                                    |            | ns   |
| TLLAX <sup>1</sup> | Address Hold After ALE Low                           | 48        |     | TCLCL-35                                    |            | ns   |
| TLLIV              | ALE Low to Valid Instr in<br>8744H<br>8044AH/8344AH  |           | 183 |   | 4TCLCL-150 | ns   |
|                    |  |           | 233 |   | 4TCLCL-100 |      |
| TLLPL              | ALE Low to PSEN Low                                  | 58        |     | TCLCL-25                                    |            | ns   |
| TPLPH              | PSEN Pulse Width<br>8744H<br>8044AH/8344AH           | 190       |     | 3TCLCL-60                                   |            | ns   |
|                    |  | 215       |     | 3TCLCL-35                                   |            | ns   |
| TPLIV              | PSEN Low to Valid Instr in<br>8744H<br>8044AH/8344AH |           | 100 |   | 3TCLCL-150 | ns   |
|                    |  |           | 125 |   | 3TCLCL-125 |      |
| TPXIX              | Input Instr Hold After PSEN                          | 0         |     | 0   |            | ns   |
| TPXIZ <sup>2</sup> | Input Instr Float After PSEN                         |           | 63  |   | TCLCL-20   | ns   |
| TPXAV <sup>2</sup> | PSEN to Address Valid                                | 75        |     | TCLCL-8                                     |            | ns   |
| TAVIV              | Address to Valid Instr in<br>8744H<br>8044AH/8344AH  |           | 267 |   | 5TCLCL-150 | ns   |
|                    |  |           | 302 |   | 5TCLCL-115 |      |
| TAZPL              | Address Float to PSEN                                | -25       |     | -25   |            | ns   |

**Notes:**

1. TLLAX for access to program memory is different from TLLAX for data memory.
2. Interfacing RUP1-44 devices with float times up to 75ns is permissible. This limited bus contention will not cause any damage to Port 0 drivers.

|      |                     |     |  |  |  |
|------|---------------------|-----|--|--|--|
| TDCH | Data Clock High     | 100 |  |  |  |
| TDCL | Data Clock Low      | 180 |  |  |  |
| TDHY | Data Clock          | 450 |  |  |  |
| TDH  | Transmit Data Delay | 140 |  |  |  |
| TDSS | Data Setup Time     | 40  |  |  |  |
| TDHS | Data Hold Time      | 40  |  |  |  |

# EXTERNAL DATA MEMORY CHARACTERISTICS

| Symbol | Parameter   | 12MHz Osc |     | Variable Clock<br>1/TCLCL = 1.2MHz to 12MHz |            | Unit |
|--------|---|-----------|-----|---|------------|------|
|        |   | Min       | Max | Min   | Max        |      |
| TRLRH  | RD Pulse Width  | 400       |     | 6TCLCL-100                                  |            | ns   |
| TWLWH  | WR Pulse Width  | 400       |     | 6TCLCL-100                                  |            | ns   |
| TLLAX  | Address Hold after ALE                                | 48        |     | TCLCL-35                                    |            | ns   |
| TRLDV  | RD Low to Valid Data in                               |           | 252 |   | 5TCLCL-165 | ns   |
| TRHDX  | Data Hold After RD                                    | 0         |     | 0   |            | ns   |
| TRHDZ  | Data Float After RD                                   |           | 97  |   | 2TCLCL-70  | ns   |
| TLLDV  | ALE Low to Valid Data In                              |           | 517 |   | 8TCLCL-150 | ns   |
| TAVDV  | Address to Valid Data In                              |           | 585 |   | 9TCLCL-165 | ns   |
| TLLWL  | ALE Low to RD or WR Low                               | 200       | 300 | 3TCLCL-50                                   | 3TCLCL+50  | ns   |
| TAVWL  | Address to RD or WR Low                               | 203       |     | 4TCLCL-130                                  |            | ns   |
| TQVWX  | Data Valid to WR Transition<br>8744H<br>8044AH/8344AH | 13        |     | TCLCL-70                                    |            | ns   |
|        |   | 23        |     | TCLCL-60                                    |            | ns   |
| TQVWH  | Data Setup Before WR High                             | 433       |     | 7TCLCL-150                                  |            | ns   |
| TWHQZ  | Data Held After WR                                    | 33        |     | TCLCL-50                                    |            | ns   |
| TRLAZ  | RD Low to Address Float                               |           | 25  |   | 25         | ns   |
| TWHLH  | RD or WR High to ALE High<br>8744H<br>8044AH/8344AH   | 33        | 133 | TCLCL-50                                    | TCLCL+50   | ns   |
|        |   | 43        | 123 | TCLCL-40                                    | TCLCL+40   | ns   |

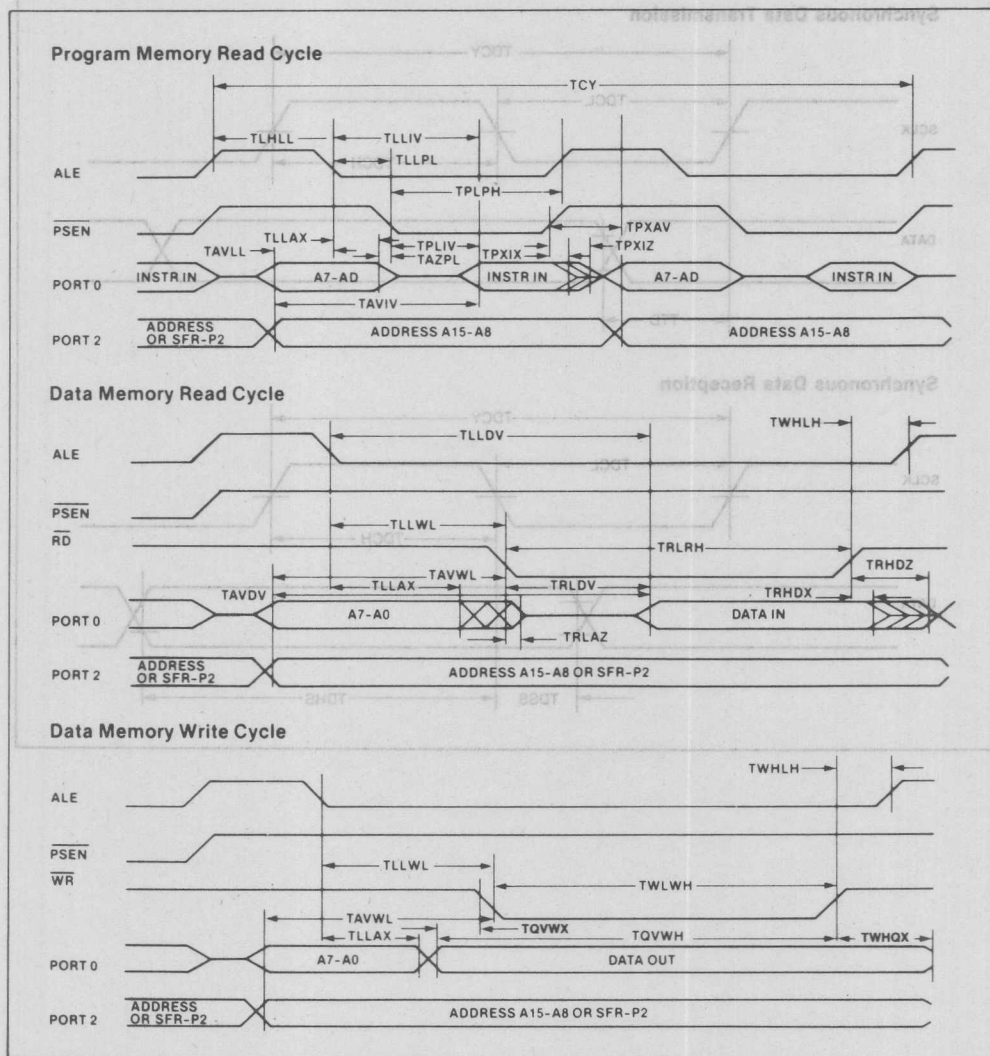
**Note 1:** TLLAX for access to program memory is different from TLLAX for access data memory.

## Serial Interface

| Symbol | Parameter           | Min | Max | Units |
|--------|---------------------|-----|-----|-------|
| TDCY   | Data Clock          | 420 |     | ns    |
| TDCL   | Data Clock Low      | 180 |     | ns    |
| TDCH   | Data Clock High     | 100 |     | ns    |
| tTD    | Transmit Data Delay |     | 140 | ns    |
| tDSS   | Data Setup Time     | 40  |     | ns    |
| tDHS   | Data Hold Time      | 40  |     | ns    |

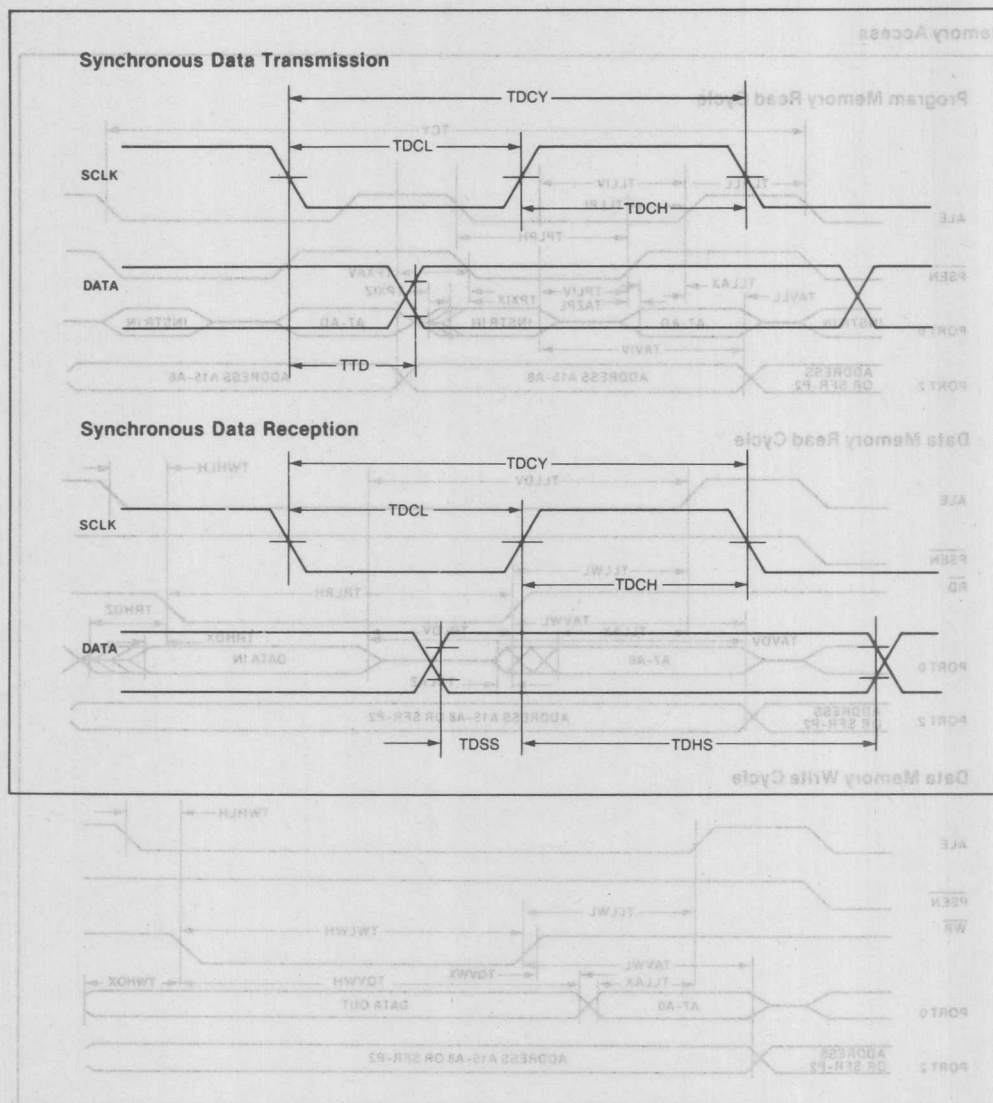
# WAVEFORMS

## Memory Access



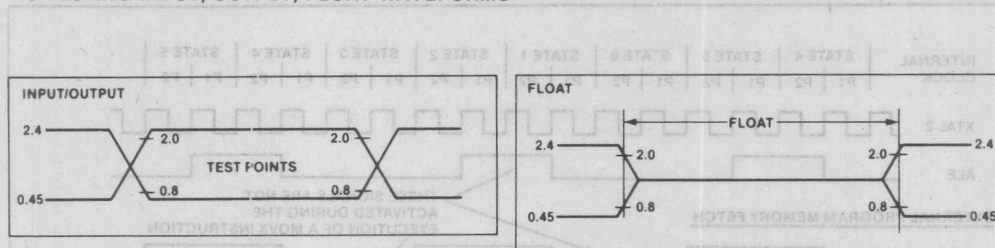
SERIAL I/O WAVEFORMS

WAVEFORMS



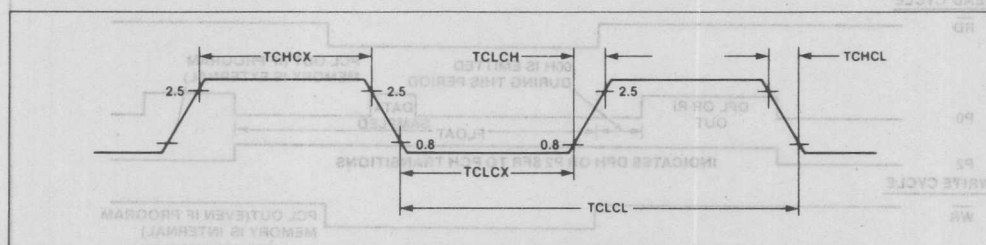


# AC TESTING INPUT, OUTPUT, FLOAT WAVEFORMS



AC testing inputs are driven at 2.4V for a logic "1" and 0.45V for a logic "0". Timing measurements are made at 2.0V for a logic "1" and 0.8V for a logic "0".

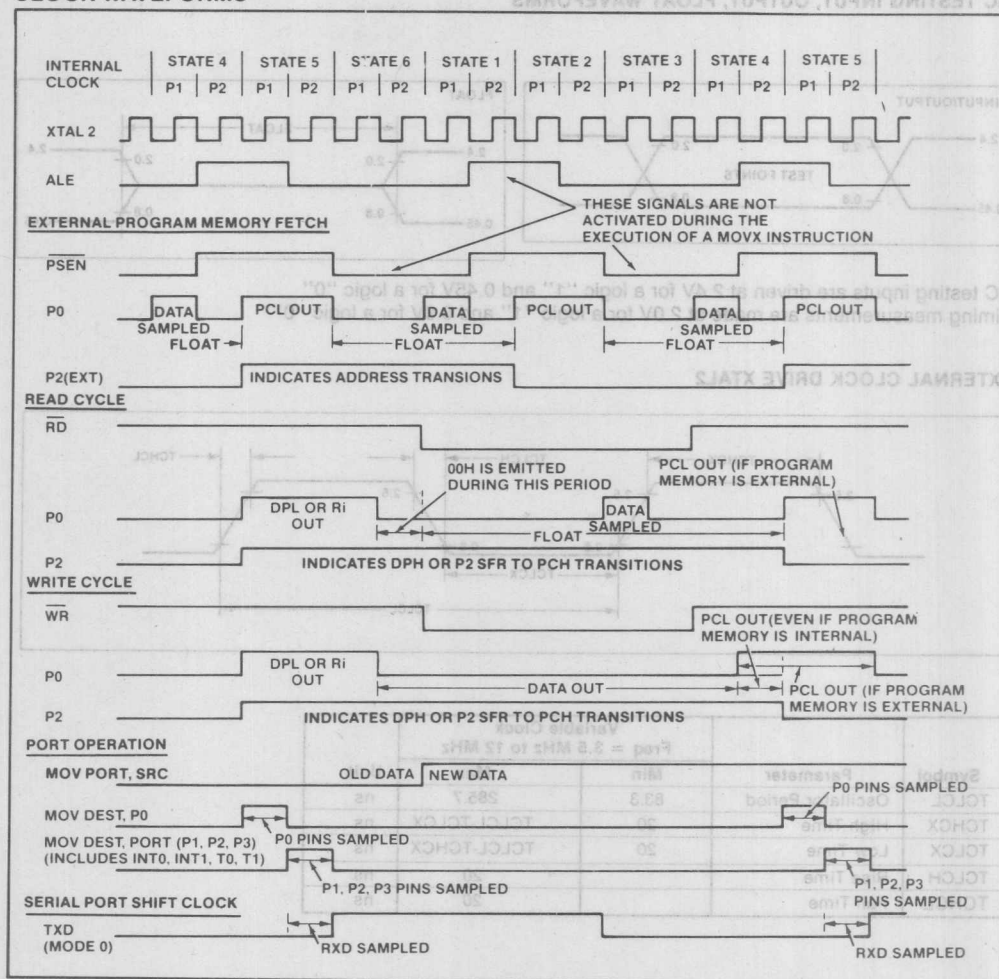
## EXTERNAL CLOCK DRIVE XTAL2



| Symbol | Parameter         | Variable Clock<br>Freq = 3.5 MHz to 12 MHz |             | Unit |
|--------|-------------------|--|-------------|------|
|        |                   | Min  | Max         |      |
| TCLCL  | Oscillator Period | 83.3                                       | 285.7       | ns   |
| TCHCX  | High Time         | 20   | TCLCL-TCLCX | ns   |
| TCLCX  | Low Time          | 20   | TCLCL-TCHCX | ns   |
| TCLCH  | Rise Time         |  | 20          | ns   |
| TCHCL  | Fall Time         |  | 20          | ns   |

This diagram indicates when signals are clocked internally. The time it takes the signals to propagate to the pins, however, ranges from 25 to 125 ns. This propagation delay is dependent on variables such as temperature and pin loading. Propagation also varies from output to output and component to component. Typically, though, (T<sub>A</sub> = 25°C, fully loaded) RD and WR propagation delays are approximately 50 ns. The other signals are typically 85 ns. Propagation delays are incorporated in the AC specifications.

# CLOCK WAVEFORMS



This diagram indicates when signals are clocked internally. The time it takes the signals to propagate to the pins, however, ranges from 25 to 125 ns. this propagation delay is dependent on variables such as temperature and pin loading. Propagation also varies from output to output and component to component. Typically though, ( $T_A = 25^\circ\text{C}$ , fully loaded) RD and WR propagation delays are approximately 50 ns. The other signals are typically 85 ns. Propagation delays are incorporated in the AC specifications.

## 8744H EPROM CHARACTERISTICS

### Erase Characteristics

Erase of the 8744H Program Memory begins to occur when the chip is exposed to light with wavelengths shorter than approximately 4,000 Ångströms. Since sunlight and fluorescent lighting have wavelengths in this range, constant exposure to these light sources over an extended period of time (about 1 week in sunlight, or 3 years in room-level fluorescent lighting) could cause unintentional erasure. If an application subjects the 8744H to this type of exposure, it is suggested that an opaque label be placed over the window.

The recommended erasure procedure is exposure to ultraviolet light (at 2537 Ångströms) to an integrated dose of at least 15 W-sec/cm<sup>2</sup> rating for 20 to 30 minutes, at a distance of about 1 inch, should be sufficient.

Erase leaves the array in an all 1s state.

### Programming the EPROM

To be programmed, the 8744H must be running with a 4 to 6 MHz oscillator. (The reason the oscillator needs to be running is that the internal bus is being used to transfer address and program data to appropriate registers.) The address of an EPROM location to be programmed is applied to Port 1 and pins P2.0-P2.3 of Port 2, while the data byte is applied to Port 0. Pins P2.4-P2.6 and PSEN should be held low, and P2.7 and RST high. (These are all TTL levels except RST, which requires 2.5V for high.) EA/VPP is held normally high, and is pulsed to +21V. While EA/VPP is at 21V, the ALE/PROG pin, which is normally being held high, is pulsed low for 50 msec. Then EA/VPP is returned to high. This is illustrated in Figure 3. Detailed timing specifications are provided in the EPROM Programming and Verification Characteristics section of this data sheet.

### Program Memory Security

The program memory security feature is developed around a "security bit" in the 8744H EPROM array. Once this "hidden bit" is programmed, electrical access to the contents of the entire program memory array becomes impossible. Activation of this feature is accomplished by programming the 8744H as described in "Programming the EPROM" with the exception that P2.6 is held at a TTL high rather than a TTL low. In addition, Port 1 and P2.0-P2.3 may be in any state. Figure 4 illustrates the security bit programming configuration. Deactivating the security feature, which again allows programmability of the EPROM, is accomplished by exposing the EPROM to ultraviolet light. This exposure, as described in "Erase Characteristics," erases the entire EPROM array. Therefore, attempted retrieval of "protected code" results in its destruction.

### Program Verification

Program Memory may be read only when the "security feature" has not been activated. Refer to Figure 5 for Program Verification setup. To read the Program Memory, the following procedure can be used. The unit must be running with a 4 to 6 MHz oscillator. The address of a Program Memory location to be read is applied to Port 1 and pins P2.0-P2.3 of Port 2. Pins P2.4-P2.6 and PSEN are held at TTL low, while the ALE/PROG, RST, and EA/VPP pins are held at TTL high. (These are all TTL levels except RST, which requires 2.5V for high.) Port 0 will be the data output lines. P2.7 can be used as a read strobe. While P2.7 is held high, the Port 0 pins float. When P2.7 is strobed low, the contents of the addressed location will appear at Port 0. External pullups (e.g., 10K) are required on Port 0 during program verification.

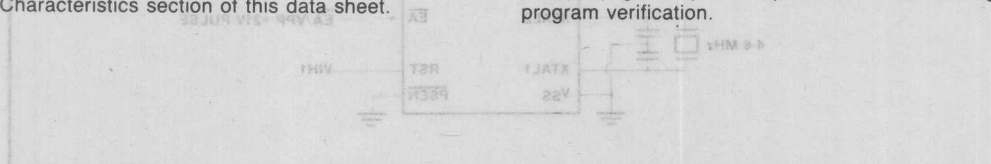


Figure 4. Security Bit Programming Configuration

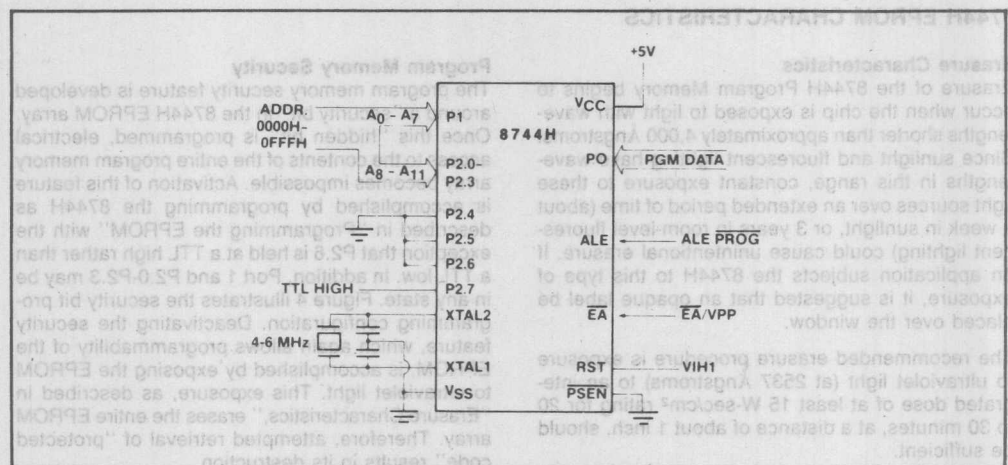


Figure 3. Programming Configuration

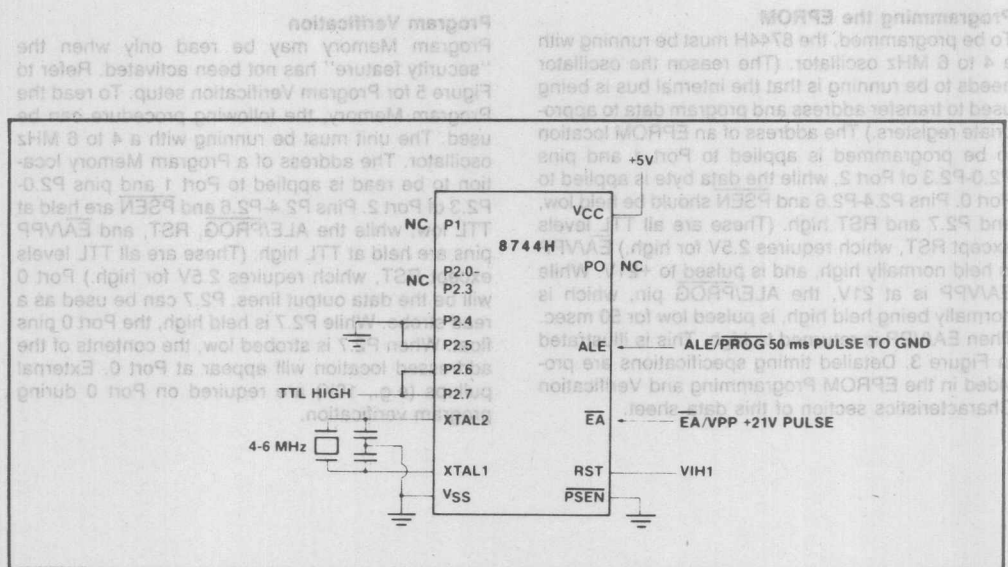


Figure 4. Security Bit Programming Configuration

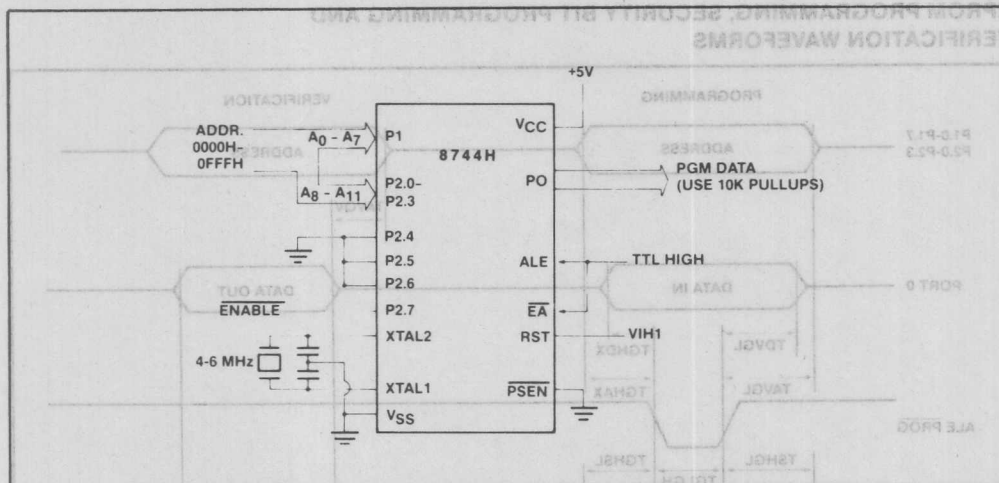


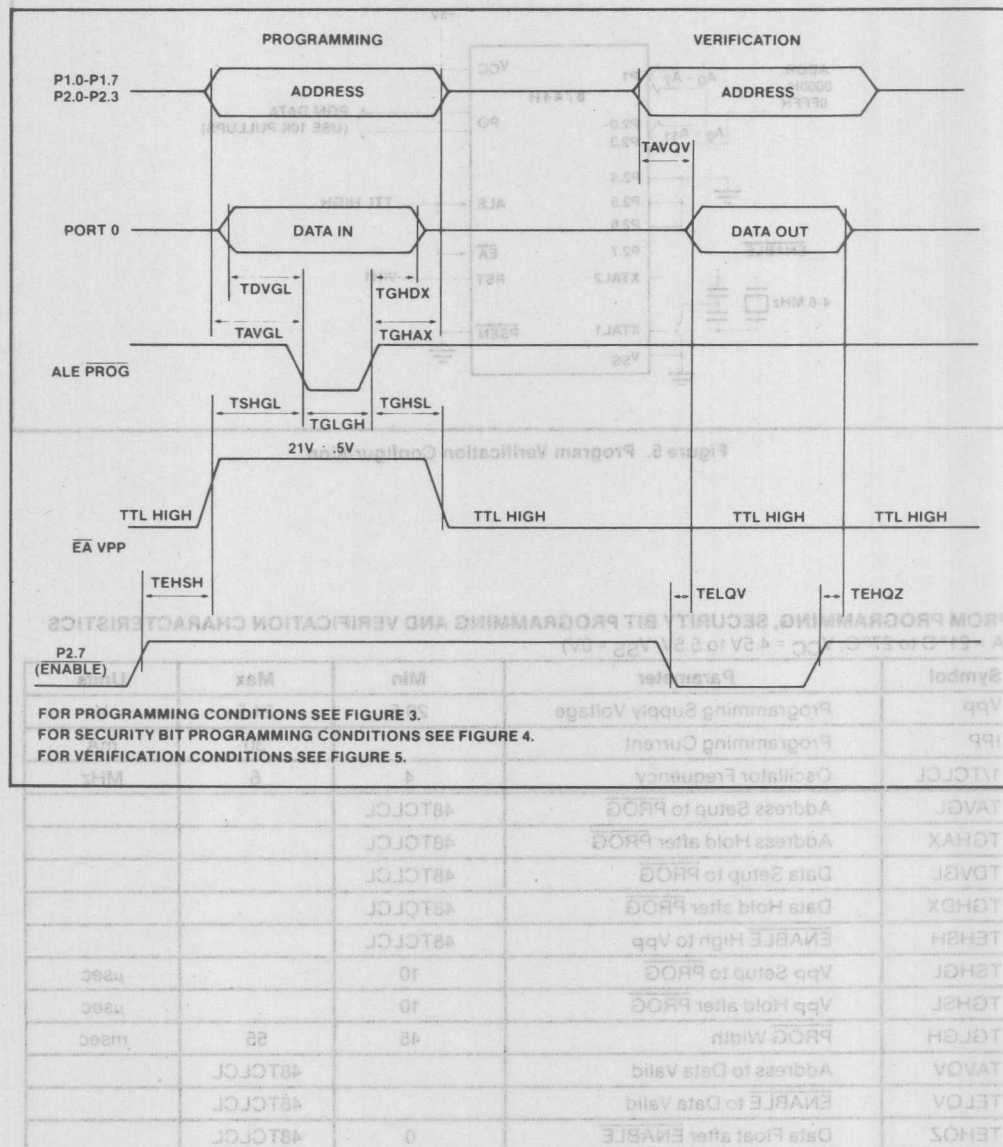
Figure 5. Program Verification Configuration

**EPROM PROGRAMMING, SECURITY BIT PROGRAMMING AND VERIFICATION CHARACTERISTICS**  
( $T_A = 21^\circ\text{C}$  to  $27^\circ\text{C}$ ,  $V_{CC} = 4.5\text{V}$  to  $5.5\text{V}$ ,  $V_{SS} = 0\text{V}$ )

| Symbol    | Parameter                             | Min     | Max     | Units           |
|-----------|---------------------------------------|---------|---------|-----------------|
| $V_{PP}$  | Programming Supply Voltage            | 20.5    | 21.5    | V               |
| IPP       | Programming Current                   |         | 30      | mA              |
| $1/TCLCL$ | Oscillator Frequency                  | 4       | 6       | MHz             |
| TAVGL     | Address Setup to $\overline{PROG}$    | 48TCLCL |         |                 |
| TGHAX     | Address Hold after $\overline{PROG}$  | 48TCLCL |         |                 |
| TDVGL     | Data Setup to $\overline{PROG}$       | 48TCLCL |         |                 |
| TGHDX     | Data Hold after $\overline{PROG}$     | 48TCLCL |         |                 |
| TEHSH     | $\overline{ENABLE}$ High to $V_{pp}$  | 48TCLCL |         |                 |
| TSHGL     | $V_{pp}$ Setup to $\overline{PROG}$   | 10      |         | $\mu\text{sec}$ |
| TGHSL     | $V_{pp}$ Hold after $\overline{PROG}$ | 10      |         | $\mu\text{sec}$ |
| TGLGH     | $\overline{PROG}$ Width               | 45      | 55      | msec            |
| TAVQV     | Address to Data Valid                 |         | 48TCLCL |                 |
| TELQV     | $\overline{ENABLE}$ to Data Valid     |         | 48TCLCL |                 |
| TEHQZ     | Data Float after $\overline{ENABLE}$  | 0       | 48TCLCL |                 |



# EPROM PROGRAMMING, SECURITY BIT PROGRAMMING AND VERIFICATION WAVEFORMS



NOVEMBER 1983

COMMUNICATIONS INTERFACE  
INTEGRATED HIGH PERFORMANCE  
MICROCONTROLLER WITH

APPLICATIONS ENGINEER  
RÉDAY 23JANUARY 1983

ORDER NUMBER 230878-001  
January 1983

© INTEL CORPORATION



ARTICLE  
REPRINT

AR-307

NOVEMBER 1983

MICROCONTROLLER WITH  
INTEGRATED HIGH PERFORMANCE  
COMMUNICATIONS INTERFACE

CHARLES YAGER  
APPLICATIONS ENGINEER

© INTEL CORPORATION

January 1985  
ORDER NUMBER 230876-001

## SUMMARY

The 8044 offers a lower cost and higher performance solution to networking microcontrollers than conventional solutions. The system cost is lowered by integrating an entire microcomputer with an intelligent HDLC/SDLC communication processor onto a single chip. The higher performance is realized by integrating two processors running concurrently on one chip; the powerful 8051 microcontroller and the Serial Interface Unit. The 8051 microcontroller is substantially off-loaded from the communication tasks when using the AUTO mode. In the AUTO mode the SIU handles many of the data link functions in hardware. The advantages of the AUTO mode are: less software is required to implement a secondary station data link, the 8051 CPU is offloaded, and the turn-around time is reduced, thus increasing the network throughput. Currently the 8044 is the only microcontroller with a sophisticated communications processor on-chip. In the future there will be more microcontrollers available following this trend.

## INTRODUCTION

Today microcontrollers are being designed into virtually every type of equipment. For the household, they are turning up in refrigerators, thermostats, burglar alarms, sprinklers, and even water softeners. At work they are found in laboratory instruments, copiers, elevators, hospital equipment, and telephones. In addition, a lot of microcomputer equipment contains more than one microcontroller. Applications using multiple microcontrollers as well, as the office and home, are now faced with the same requirements that laboratory instruments were faced with 12 years ago — they need to connect them together and have them communicate. This need was satisfied in the laboratory with the IEEE-488 General Purpose Instrumentation Bus (GPIB). However, GPIB does not meet the current design objectives for networking microcontrollers.

Today there are many communications schemes and protocols available; some of the popular ones are GPIB, Async, HDLC/SDLC, and Ethernet. Common design objectives of today's networks are: low cost, reliable, efficient throughput, and expandable. In examining available solutions, GPIB does not meet these design objectives; first, the cable is too expensive (parallel communications), second, it can only be used over a limited distance (20 meters), and third, it can only handle a limited number of stations. For general networking, serial communications is preferable because of lower cable costs and higher reliability (fewer connections). While Ethernet provides very high performance, it is more of a system backbone rather than a microcontroller interconnect. Async, on the other hand, is inexpensive but it is not an efficient protocol for data block or file transfers. Even with some new modifications such as a 9 bit protocol for addressing, important functions such as

acknowledgements, error checking/recovery, and data transparency are not standardized nor supported by available data comm chips.

SDLC, Synchronous Data Link Control, meets the requirements for communications link design. The physical medium can be used on two or four wire twisted pair with inexpensive transceivers and connectors. It can also be interfaced through modems, which allows it to be used on broadband networks, leased or switched telephone lines. VLSI controllers have been available from a number of vendors for years; higher performance and more user friendly SDLC controllers continue to appear. SDLC has also been designed to be very reliable. A 16 bit CRC checks the integrity of the received data, while frame numbering and acknowledgements are also built in. Using SDLC, up to 254 stations can be uniquely addressed, while HDLC addressing is unlimited. If an RS-422 only requires a single +5 volt power supply.

What will the end user pay for the added value provided by communications? The cost of the communications hardware is not the only additional cost. There will be performance degradation in the main application because the microcontroller now has additional tasks to perform. There are two extremes to the cost of adding communication capability. One could spend very little by adding an I/O port and have the CPU handle everything from the baud rate to the protocol. Of course the main application would be idle while the CPU was communicating. The other extreme would be to add another microcontroller to the system dedicated to communications. This communications processor could interface to the main CPU through a high speed parallel link or dual port RAM. This approach would maintain system performance, but it would be costly.

### Adding HDLC/SDLC Networking Capability

Figure 1 shows a microcomputer system with a conventional HDLC/SDLC communications solution. The additional hardware needed to realize the conventional design is: an HDLC/SDLC communication chip, additional ROM for the communication software, part of an interrupt controller, a baud rate generator, a phase locked loop, NRZI encoded/decoder, and a cable driver locked loop are used when the transmitter does not send the clock on a separate line from the data (i.e., over telephone lines, or two wire cable). the NRZI encoder/decoder is used in HDLC/SDLC to combine the clock into the data line. A phase locked loop is used to recover the clock from the data line.

The majority of the available communication chips provide a limited number of data link control functions. Most of them will handle Zero Bit Insertion/Deletion (ZBI/D), Flags, Aborts, Automatic



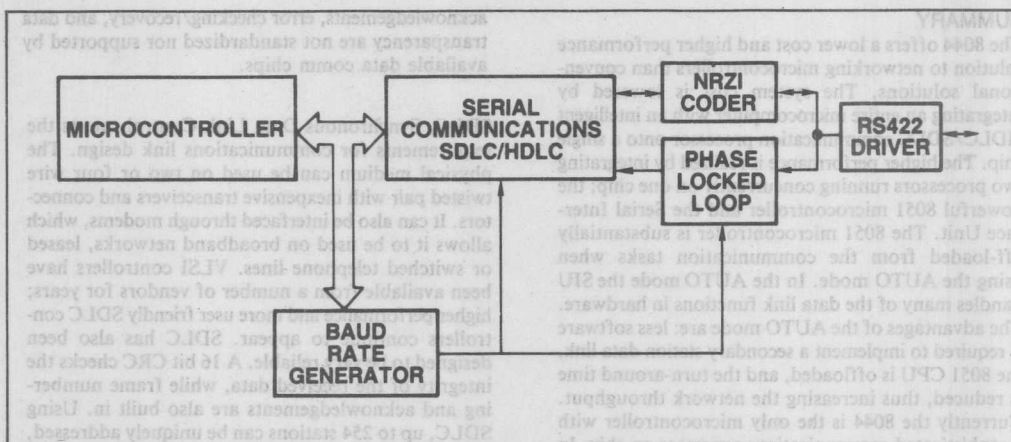


Figure 1. Conventional microcontroller networking solution

address recognition, and CRC generation and checking. It is the CPU's responsibility to manage link access, command recognition and response, acknowledgements and error recovery. Handling these tasks can take a lot of CPU time. In addition, servicing the transmission and reception of data bytes can also be very time consuming depending on the method used.

Using a DMA controller can increase the overall system performance, since it can transfer a block of data in fewer clock cycles than a CPU. In addition, the CPU and the DMA controller can multiplex their access to the bus so that both can be running at virtually the same time. However, both the DMA controller and the CPU are sharing the same bus, therefore, neither one get to utilize 100% of the bus bandwidth. Microcontrollers available today do not support DMA, therefore, they would have to use interrupts, since polling is unacceptable in a multitasking environment.

In an interrupt driven, the CPU has overhead in addition to servicing the interrupt. During each interrupt request the CPU has to save all of the important registers, transfer a byte, update pointers and counters, then restore all of its registers. At low bit rates this overhead may be insignificant. However, the percentage of overhead increases linearly with the bit rate. At high bit rates this overhead would consume all of the CPU's time. There is another nuisance factor associated with interrupt driven systems, interrupt latency. Too much interrupt latency will cause data to be lost from underrun and overrun errors.

The additional hardware necessary to implement the communications solution, as shown in Figure 1, would

require 1 LSI chip and about 10 TTL chips. The cost of CPU throughput degradation can be even greater. The percentage of time the CPU has to spend servicing the communication tasks can be anywhere from 10-100%, depending on the serial bit rate. These high costs will prevent consumer acceptance of networking microcomputer equipment.

**A Highly Integrated, High Performance Solution**  
The 8044 reduces the cost of networking microcontrollers without compromising performance. It contains all of the hardware components necessary to implement a microcomputer system with communications capability, plus it reduces the CPU and software overhead of implementing HDLC/SDLC. Figure 2 shows a functional block diagram of the 8044.

The 8044 integrates the powerful 8051 microcontroller with an intelligent Serial Interface Unit to provide a single chip solution which efficiently implements a distributed processing or distributed control system. The microcontroller is a self sufficient unit containing ROM, RAM, ALU and its own peripherals. The 8044's architecture and instruction set are identical to the 8051's. The Serial Interface Unit (SIU) uses bit synchronous HDLC/SDLC protocol and can communicate at bit rates up to 2.4 Mbps, externally clocked, or up to 375 Kbps using the on-chip digital phase locked loop. The SIU contains its own processor, which operates concurrently with the microcontroller.

The CPU and the SIU, in the 8044, interface through 192 bytes of dual port RAM. There is no hardware arbitration in the dual port RAM. Both processor's memory access cycles are interlaced; each processor has access every other clock cycle. Therefore, there



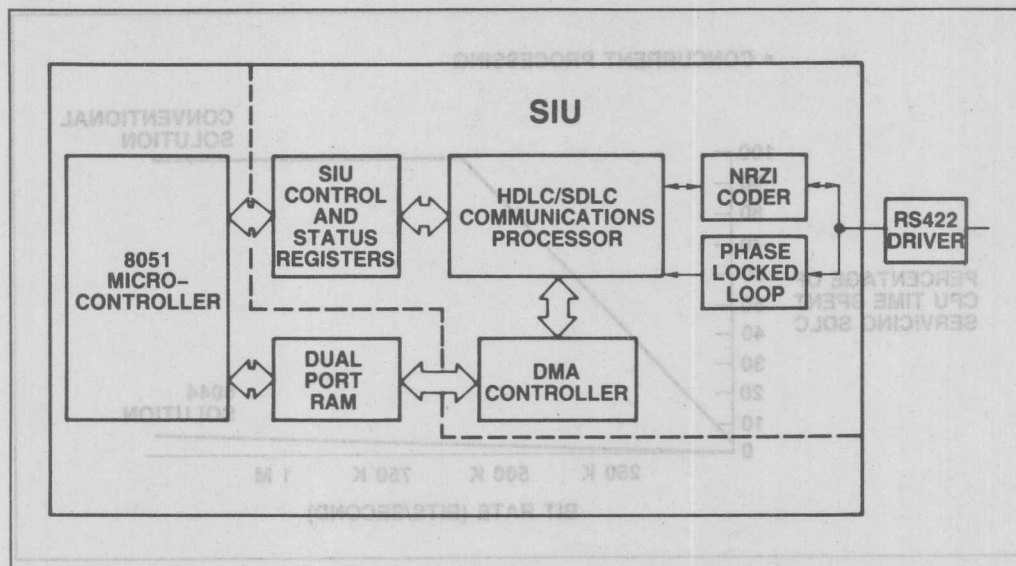


Figure 2. 8044 single chip microcontroller networking solution

is no throughput loss in either processor as a result of the dual port RAM, and execution times are deterministic. Since this has always been the method for memory access on the 8051 microcontroller, 8051 programs have the same execution time in the 8044.

By integrating all of the communication hardware onto the 8051 microcontroller, the hardware cost of the system is reduced. Now several chips have been integrated into a single chip. This means that the system power is reduced, P.C. board space is reduced, inventory and assembly is reduced, and reliability is improved. The improvement in reliability is a result of fewer chips and interconnections on the P.C. board.

As mentioned before, there can be two extremes in a design which adds communications to the microcomputer system. The 8044 solution uses the high end extreme. The SIU on the 8044 contains its own processor which communicates with the 8051 processor through dual port RAM and control/status registers. While the SIU is not a totally independent communications processor, it substantially offloads the 8051 processor from the communication tasks.

The DMA on the 8044 is dedicated to the SIU. It cannot access external RAM. By having a DMA controller in the SIU, the 8051 CPU is offloaded. As a result of the dual port RAM design, the DMA does not share the running at full speed while the frames are being

transmitted or received. Also, the nuisance of overrun and underrun errors is totally eliminated since the dedicated DMA controller is guaranteed to meet the maximum data rates. Having a dedicated DMA controller means that the serial channel interrupt can be the lowest priority, thus allowing the CPU to have higher priority real time interrupts.

Figure 3 shows a comparison between the conventional and the 8044 solution on the percentage of time the CPU must spend sending data. This diagram was derived by assuming a 64 byte information frame is being transmitted repeatedly. The conventional solution is interrupt driven, and each interrupt service routine is assumed to take about 15 instructions with a 1  $\mu$ sec instruction cycle time. At 533 Kbps, an interrupt would occur every 15  $\mu$ sec. Thus, the CPU becomes completely dedicated to servicing the serial communications. The conventional design could not support bit rates higher than this because of underruns and overruns. For the 8044 to repeatedly send 64 byte frames, it simply has to reinitialize the DMA controller. As a result, the 8044 can support bit rates up to 2.4 Mbps.

Some of the other communications tasks the CPU has to perform, such as link access, command recognition/response, and acknowledgements, are performed automatically by the SIU in a mode called "AUTO." The combination of the dedicated DMA controller and the AUTO mode, substantially offload

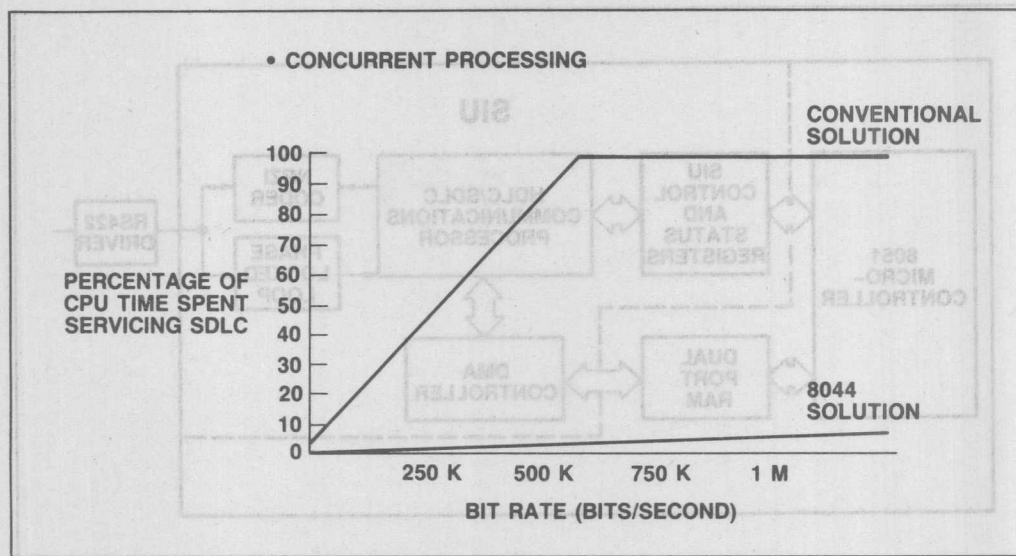


Figure 3. SIU offloads CPU

the CPU, thus allowing it to devote more of its power to other tasks.

#### 8044's Auto Mode

In the AUTO mode the SIU implements in hardware a subset of the SDLC protocol such that it responds to many SDLC commands without CPU intervention. All AUTO mode responses to the primary station conform to IBM's SDLC definition. In the AUTO mode the 8044 can only be a secondary station operating in SDLC specified "Normal Response Mode." Normal Response Mode means that the secondary station can not transmit unless it is polled by the primary station. The SIU in the AUTO mode can recognize and respond to the following SDLC commands without CPU intervention: I (Information), RR (Receive Ready), RNR (Receive Not Ready), REJ (Reject), and for loop mode UP (Unnumbered Poll). The SIU can generate the following responses without CPU intervention: I, RR, and RNR. In addition, the SIU manages Ns and Nr in the control field. If it detects an error in either Ns or Nr, it interrupts the CPU for error recovery.

How does the SIU know what responses to send to the primary? It uses two status bits which are set by the CPU. The two bits are TBF (Transmit Buffer Full) and RBP (Receive Buffer Protect). TBF indicates that the CPU wants to send data, and RBP indicates that the receive data buffer is full. Table 1 shows the responses the SIU will send based on these two status bits. This is an innovative approach to communication design. The CPU in the 8044 with one instruction

can directly set a bit which communicates to the primary what its transmit and receive buffering status is.

When the CPU wants to send a frame, it loads the transmit buffer with the data, loads the starting address and the count of the data into the SIU, then sets TBF to transmit the frame. The SIU waits for the primary station to poll it with a RR command. After the SIU is polled, it automatically sends the information frame to the primary with the proper control field. The SIU then waits for a positive acknowledgement from the primary before incrementing the Ns field and interrupting the CPU for more data. If a negative acknowledgement is received, the SIU automatically retransmits the frame.

When the 8044 is ready to receive information, the CPU loads the receive buffer starting address and the buffer length into the SIU, then enables the receiver. When a valid information frame with the correct address and CRC is received, the SIU will increment the Nr field, disable the receiver and interrupt the CPU indicating that a good I frame has been received. The CPU then sets RBP, reenables the receiver and processes the received data. By enabling the receiver with RBP set, the SIU will automatically respond to polls with a Receive Not Ready, thus keeping the link moving rather than timing out the primary from a disabled receiver, or interrupting the CPU with another poll before it has processed the data. After the data has been processed, the CPU clears RBP, returning to the Receive Ready responses.

Table 1. SIU's automatic responses in auto mode

| STATUS BITS |     | RESPONSE                |
|-------------|-----|-------------------------|
| TBF         | RBP |                         |
| 0           | 0   | (RR) Receive ready      |
| 0           | 1   | (RNR) Receive not ready |
| 1           | 0   | (I) Information         |
| 1           | 1   | (I) Information         |

SDLC communications can be broken up into four states: Logical Disconnect State, Initialization State, Frame Reject State, and Information Transfer State. Data can only be transferred in the Information Transfer State. More than 90% of the time a station will be in the Information Transfer State, which is where the SIU can run autonomously. In the other states, where error recovery, online/offline, and initialization takes place, the CPU manages the protocol.

In the Information Transfer State there are three common events which occur as illustrated in Figure 4, they are: 1) the primary polls the secondary and the secondary is ready to receive but has nothing to send, 2) the primary sends the secondary information, and 3) the secondary sends information to the primary. Figures 5, 6, and 7 compare the functions the conventional design and the 8044 must execute in order to respond to the primary for the cases in Figure 4.

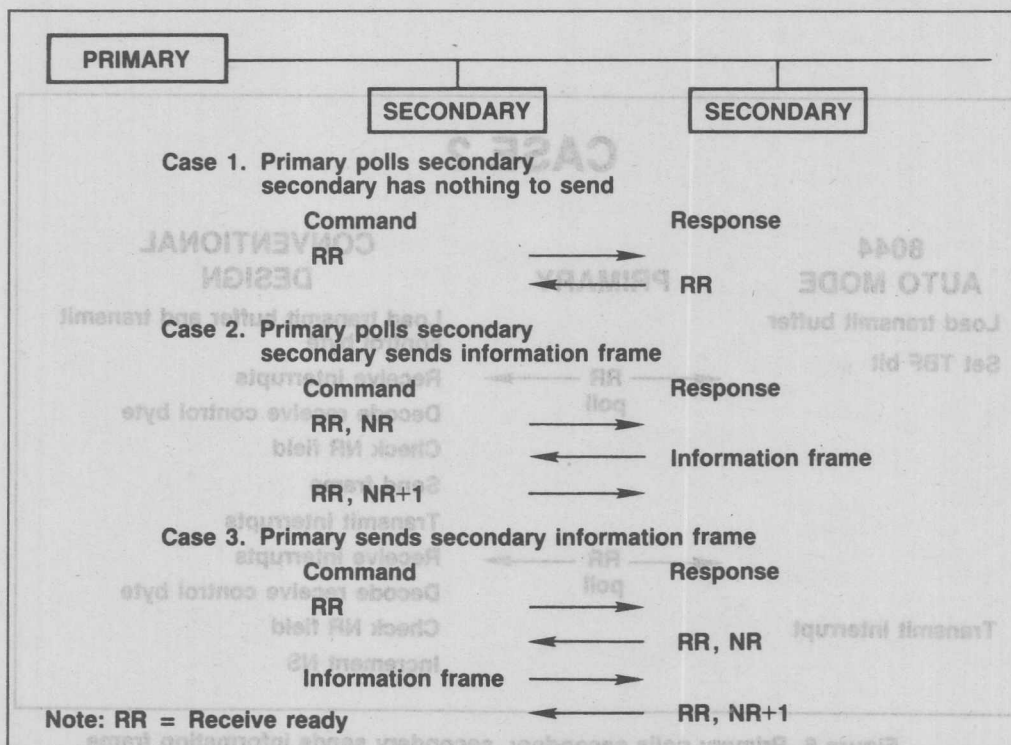


Figure 4. SDLC commands and responses in the information transfer state

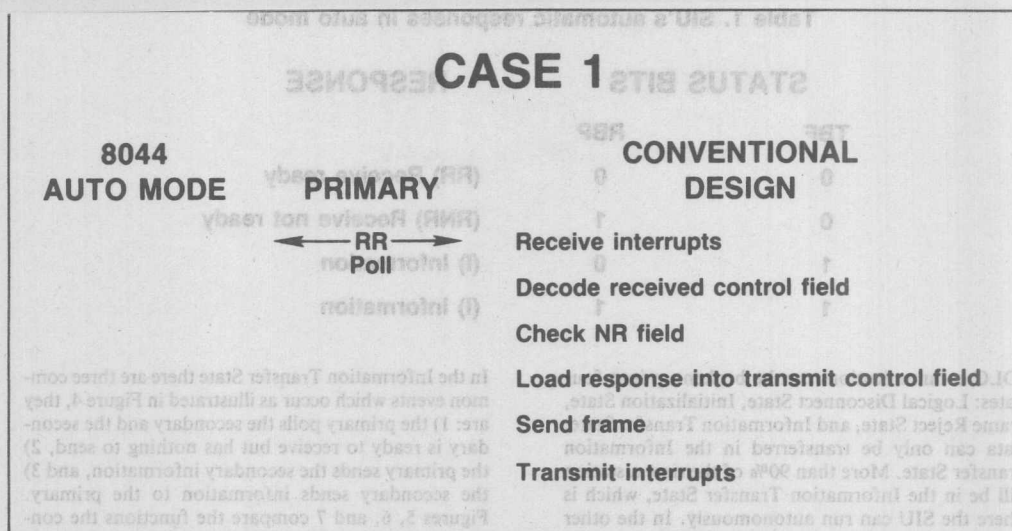


Figure 5. Primary polls secondary, secondary has nothing to send

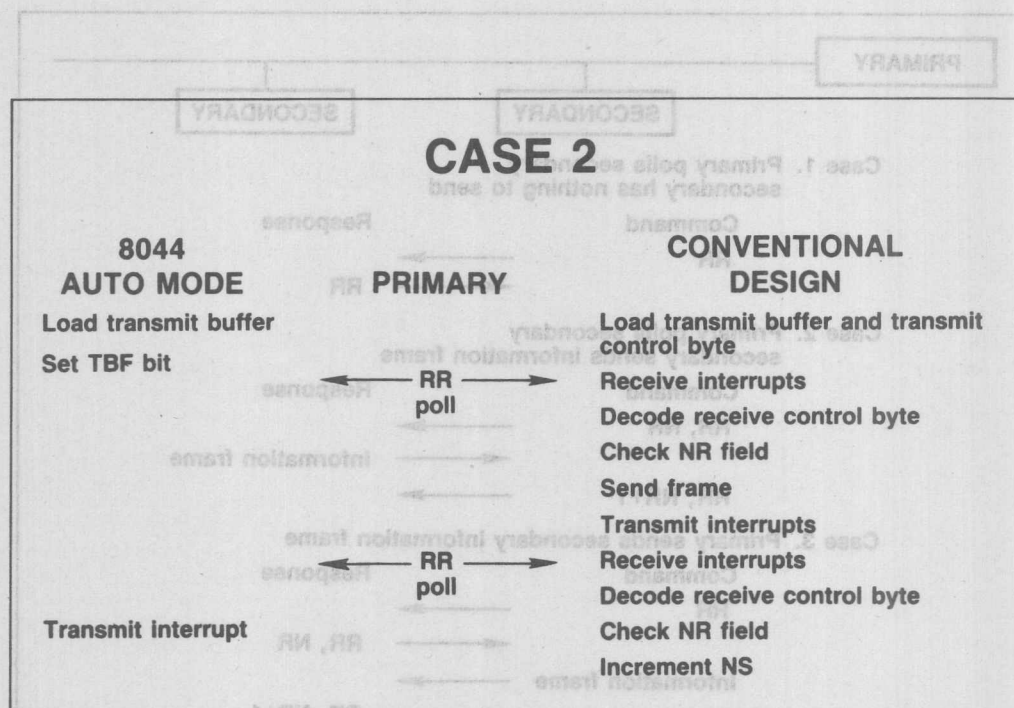


Figure 6. Primary polls secondary, secondary sends information frame



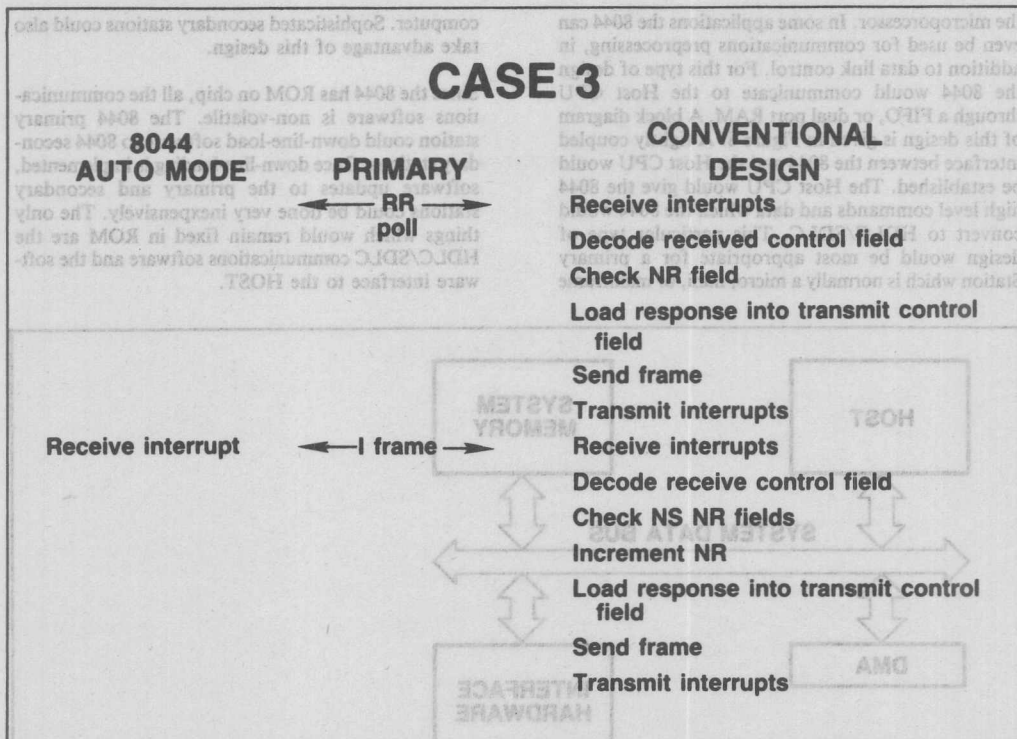


Figure 7. Primary sends information frame to secondary

Using case 1 as an example, the conventional design first gets receive interrupts bringing the data from the SDLC comm chip into memory. The CPU must then decode the command in the control field and determine the response. In addition, it must check the Nr field for any pending acknowledgements. The CPU loads the transmit buffer with the appropriate address and control field, then transmits the frame. When the 8044 receives this frame in AUTO mode, the CPU never gets an interrupt because the SIU handles the entire frame reception and response automatically.

In SDLC networks, when there is no information transfers, case 1 is the activity on the line. Typically this is 80% of the network traffic. The CPU in the conventional design would constantly be getting interrupts and servicing the communications tasks, even when it has nothing to send or receive. On the other hand, the 8044 CPU would only get involved in communicating when it has data to send or receive.

Having the SIU implement a subset of the SDLC protocol in hardware not only offloads the CPU, but it also improves the throughput on the network. The

most critical parameter for calculating throughput on any high speed network is the station turnaround time; the time it takes a station to respond after receiving a frame. Since the 8044 handles all of the commands and responses of the Information Transfer State in hardware, the turnaround time is much faster than handling it in software, hence a higher throughput.

#### 8044's Flexible Mode

In the "NON-AUTO" mode or Flexible mode, the SIU does not recognize or respond to any commands, nor does it manage acknowledgements, which means the CPU must handle link access, command recognition/response, acknowledgements and error recovery by itself. The Flexible mode allows the 8044 to have extended address fields and extended control fields, thus providing HDLC support. In the Flexible mode the 8044 can operate as a primary station, since it can transmit without being polled.

#### Front End Communications Processor

The 8044 can also be used as an intelligent HDLC/SDLC front end for a microprocessor, capable of extensively off-loading link control functions for



the microprocessor. In some applications the 8044 can even be used for communications preprocessing, in addition to data link control. For this type of design the 8044 would communicate to the Host CPU through a FIFO, or dual port RAM. A block diagram of this design is given in Figure 8. A tightly coupled interface between the 8044 and the Host CPU would be established. The Host CPU would give the 8044 high level commands and data which the 8044 would convert to HDLC/SDLC. This particular type of design would be most appropriate for a primary Station which is normally a micro, mini, or mainframe

computer. Sophisticated secondary stations could also take advantage of this design.

Since the 8044 has ROM on chip, all the communications software is non-volatile. The 8044 primary station could down-line-load software to 8044 secondary stations. Once down-line-loading is implemented, software updates to the primary and secondary stations could be done very inexpensively. The only things which would remain fixed in ROM are the HDLC/SDLC communications software and the software interface to the HOST.

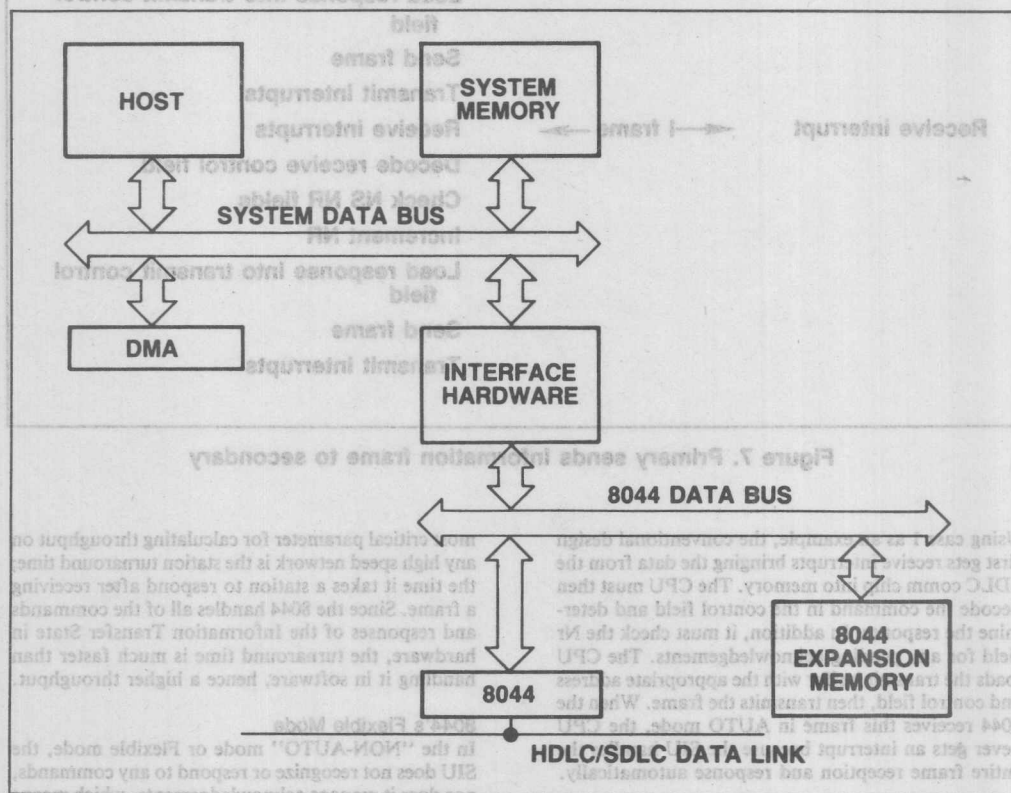


Figure 8. 8044 front end processor

Front End Communications Processor  
The 8044 can also be used as an intelligent HDLC/SDLC front end for microprocessor, capable of extensively off-loading link control functions for

In SDLC networks, when there is no information transfer, case 1 is the activity on the line. This is 80% of the network traffic. The CPU in the conventional design would constantly be getting interrupts and servicing the communications tasks, even when it has nothing to send or receive. On the other hand, the 8044 CPU would only get involved in communicating when it has data to send or receive.

Having the 8044 implement a subset of the SDLC protocol in hardware not only offloads the CPU, but it also improves the throughput on the network. The

# Systems for Electrically Noisy Environments

|       |   |
|-------|---|
| 22-2  | SYMPOMS OF NOISE PROBLEMS                   |
| 22-3  | TYPES AND SOURCES OF ELECTRICAL NOISE       |
| 22-3  | Supply Line Transients                      |
| 22-3  | EMP and RFI                                 |
| 22-3  | ESD   |
| 22-3  | Ground Noise                                |
| 22-3  | "RADIATED" AND "CONDUCTED" NOISE            |
| 22-4  | SIMULATING THE ENVIRONMENT                  |
| 22-4  | TYPES OF FAILURES AND FAILURE MECHANISMS    |
| 22-5  | THE GAME PLAN                               |
| 22-5  | CURRENT LOOPS                               |
| 22-5  | SHIELDING                                   |
| 22-5  | Shielding Against Capacitive Coupling       |
| 22-5  | Shielding Against Inductive Coupling        |
| 22-5  | RF Shielding                                |
| 22-10 | GROUND                                      |
| 22-10 | Safety Ground                               |
| 22-11 | Signal Ground                               |
| 22-12 | Practical Grounding                         |
| 22-13 | Braided Cable                               |
| 22-14 | POWER SUPPLY DISTRIBUTION AND DECOUPLING    |
| 22-15 | Selecting the Value of the Decoupling Cap   |
| 22-16 | The Case for On-Board Voltage Regulation    |
| 22-16 | RECOVERING GRACEFULLY FROM A SOFTWARE UPSET |
| 22-16 | SPECIAL PROBLEM AREAS                       |
| 22-16 | ESD   |
| 22-19 | The Automotive Environment                  |
| 22-21 | PARTING THOUGHTS                            |
| 22-22 | REFERENCES                                  |

22

# Designing Microcontroller Systems for Electrically Noisy Environments

Design Considerations

## Contents

|   |              |
|---|--------------|
| <b>SYMPTOMS OF NOISE PROBLEMS . . . . .</b>                       | <b>22-2</b>  |
| <b>TYPES AND SOURCES OF ELECTRICAL NOISE</b>                      |              |
| Supply Line Transients . . . . .                                  | 22-2         |
| EMP and RFI . . . . .   | 22-2         |
| ESD . . . . .   | 22-3         |
| Ground Noise . . . . .  | 22-3         |
| <b>"RADIATED" AND "CONDUCTED" NOISE . .</b>                       | <b>22-3</b>  |
| <b>SIMULATING THE ENVIRONMENT . . . . .</b>                       | <b>22-4</b>  |
| <b>TYPES OF FAILURES AND FAILURE<br/>    MECHANISMS . . . . .</b> | <b>22-4</b>  |
| <b>THE GAME PLAN . . . . .</b>                                    | <b>22-5</b>  |
| <b>CURRENT LOOPS . . . . .</b>                                    | <b>22-5</b>  |
| <b>SHIELDING . . . . .</b>  | <b>22-6</b>  |
| Shielding Against Capacitive Coupling . . . . .                   | 22-6         |
| Shielding Against Inductive Coupling . . . . .                    | 22-6         |
| RF Shielding . . . . .  | 22-9         |
| <b>GROUND . . . . .</b>   | <b>22-10</b> |
| Safety Ground . . . . .   | 22-10        |
| Signal Ground . . . . .   | 22-11        |
| Practical Grounding . . . . .                                     | 22-12        |
| Braided Cable . . . . .   | 22-13        |
| <b>POWER SUPPLY DISTRIBUTION AND<br/>DECOUPLING . . . . .</b>     | <b>22-14</b> |
| Selecting the Value of the Decoupling Cap . . . . .               | 22-15        |
| The Case for On-Board Voltage Regulation . . . . .                | 22-16        |
| <b>RECOVERING GRACEFULLY FROM A SOFTWARE<br/>UPSET . . . . .</b>  | <b>22-16</b> |
| <b>SPECIAL PROBLEM AREAS . . . . .</b>                            | <b>22-18</b> |
| ESD . . . . .   | 22-18        |
| The Automotive Environment . . . . .                              | 22-19        |
| <b>PARTING THOUGHTS . . . . .</b>                                 | <b>22-21</b> |
| <b>REFERENCES . . . . .</b>                                       | <b>22-22</b> |

Digital circuits are often thought of as being immune to noise problems, but really they're not. Noises in digital systems produce software upsets: program jumps to apparently random locations in memory. Noise-induced glitches in the signal lines can cause such problems, but the supply voltage is more sensitive to glitches than the signal lines.

Severe noise conditions, those involving electrostatic discharges, or as found in automotive environments, can do permanent damage to the hardware. Electrostatic discharges can blow a crater in the silicon. In the automotive environment, in ordinary operation, the "12V" power line can show + and -400V transients.

This Application Note describes some electrical noises and noise environments. Design considerations, along the lines of PCB layout, power supply distribution and decoupling, and shielding and grounding techniques, that may help minimize noise susceptibility are reviewed. Special attention is given to the automotive and ESD environments.

## Symptoms of Noise Problems

Noise problems are not usually encountered during the development phase of a microcontroller system. This is because benches rarely simulate the system's intended environment. Noise problems tend not to show up until the system is installed and operating in its intended environment. Then, after a few minutes or hours of normal operation the system finds itself someplace out in left field. Inputs are ignored and outputs are gibberish. The system may respond to a reset, or it may have to be turned off physically and then back on again, at which point it commences operating as though nothing had happened. There may be an obvious cause, such as an electrostatic discharge from somebody's finger to a keyboard or the upset occurs every time a copier machine is turned on or off. Or there may be no obvious cause, and nothing the operator can do will make the upset repeat itself. But a few minutes, or a few hours, or a few days later it happens again.

One symptom of electrical noise problems is randomness, both in the occurrence of the problem and in what the system does in its failure. All operational upsets that occur at seemingly random intervals are not necessarily caused by noise in the system. Marginal VCC, inadequate decoupling, rarely encountered software conditions, or timing coincidences can produce upsets that seem to occur randomly. On the other hand, some noise sources can produce upsets downright periodically. Nevertheless, the more difficult it is to characterize an upset as to cause and effect, the more likely it is to be a noise problem.

## Types and Sources of Electrical Noise

The name given to electrical noises other than those that are inherent in the circuit components (such as thermal noise) is EMI: electromagnetic interference. Motors, power switches, fluorescent lights, electrostatic discharges, etc., are sources of EMI. There is a veritable alphabet soup of EMI types, and these are briefly described below.

### SUPPLY LINE TRANSIENTS

Anything that switches heavy current loads onto or off of AC or DC power lines will cause large transients in these power lines. Switching an electric typewriter on or off, for example, can put a 1000V spike onto the AC power lines.

The basic mechanism behind supply line transients is shown in Figure 1. The battery represents any power source, AC or DC. The coils represent the line inductance between the power source and the switchable loads R1 and R2. If both loads are drawing current, the line current flowing through the line inductance establishes a magnetic field of some value. Then, when one of the loads is switched off, the field due to that component of the line current collapses, generating transient voltages,  $v = L(di/dt)$ , which try to maintain the current at its original level. That's called an "inductive kick." Because of contact bounce, transients are generated whether the switch is being opened or closed, but they're worse when the switch is being opened.

An inductive kick of one type or another is involved in most line transients, including those found in the automotive environment. Other mechanisms for line transients exist, involving noise pickup on the lines. The noise voltages are then conducted to a susceptible circuit right along with the power.

### EMP AND RFI

Anything that produces arcs or sparks will radiate electromagnetic pulses (EMP) or radio-frequency interference (RFI).

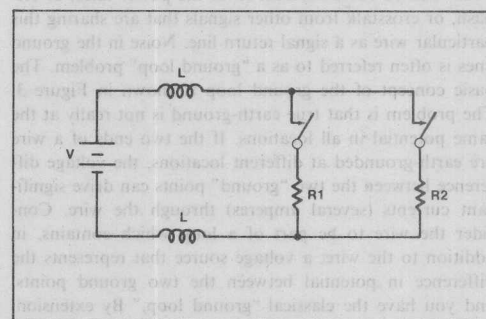


Figure 1. Supply Line Transients



Spark discharges have probably caused more software upsets in digital equipment than any other single noise source. The upsetting mechanism is the EMP produced by the spark. The EMP induces transients in the circuit, which are what actually cause the upset.

Arcs and sparks occur in automotive ignition systems, electric motors, switches, static discharges, etc. Electric motors that have commutator bars produce an arc as the brushes pass from one bar to the next. DC motors and the "universal" (AC/DC) motors that are used to power hand tools are the kinds that have commutator bars. In switches, the same inductive kick that puts transients on the supply lines will cause an opening or closing switch to throw a spark.

### ESD

Electrostatic discharge (ESD) is the spark that occurs when a person picks up a static charge from walking across a carpet, and then discharges it into a keyboard, or whatever else can be touched. Walking across a carpet in a dry climate, a person can accumulate a static voltage of 35kV. The current pulse from an electrostatic discharge has an extremely fast risetime — typically, 4A/nsec. Figure 2 shows ESD waveforms that have been observed by some investigators of ESD phenomena.

It is enlightening to calculate the  $L(di/dt)$  voltage required to drive an ESD current pulse through a couple of inches of straight wire. Two inches of straight wire has about 50nH of inductance. That's not very much, but using 50nH for L and 4A/nsec for  $di/dt$  gives an  $L(di/dt)$  drop of about 200V. Recent observations by W.M. King suggest even faster risetimes (Figure 2B) and the occurrence of multiple discharges during a single discharge event.

Obviously, ESD-sensitivity needs to be considered in the design of equipment that is going to be subjected to it, such as office equipment.

### GROUND NOISE

Currents in ground lines are another source of noise. These can be 60Hz currents from the power lines, or RF hash, or crosstalk from other signals that are sharing this particular wire as a signal return line. Noise in the ground lines is often referred to as a "ground loop" problem. The basic concept of the ground loop is shown in Figure 3. The problem is that true earth-ground is not really at the same potential in all locations. If the two ends of a wire are earth-grounded at different locations, the voltage difference between the two "ground" points can drive significant currents (several amperes) through the wire. Consider the wire to be part of a loop which contains, in addition to the wire, a voltage source that represents the difference in potential between the two ground points, and you have the classical "ground loop." By extension, the term is used to refer to any unwanted (and often unexpected) currents in a ground line.

### "Radiated" and "Conducted" Noise

Radiated noise is noise that arrives at the victim circuit in the form of electromagnetic radiation, such as EMP and RFI. It causes trouble by inducing extraneous voltages in the circuit. Conducted noise is noise that arrives at the victim circuit already in the form of an extraneous voltage, typically via the AC or DC power lines.

One defends against radiated noise by care in designing layouts and the use of effective shielding techniques. One defends against conducted noise with filters and suppress-

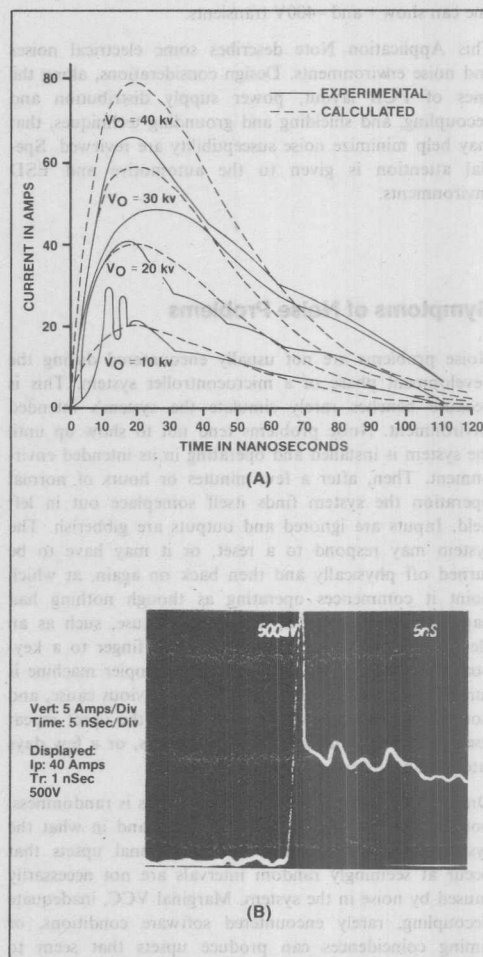


Figure 2. Waveforms of Electrostatic Discharge Currents From a Hand-Held Metallic Object



sors, although layouts and grounding techniques are important here, too.

## Simulating the Environment

Addressing noise problems after the design of a system has been completed is an expensive proposition. The ill will generated by failures in the field is not cheap either. It's cheaper in the long run to invest a little time and money in learning about noise and noise simulation equipment, so that controlled tests can be made on the bench as the design is developing.

Simulating the intended noise environment is a two-step process: First you have to recognize what the noise environment is, that is, you have to know what kinds of electrical noises are present, and which of them are going to cause trouble. Don't ignore this first step, because it's important. If you invest in an induction coil spark generator just because your application is automotive, you'll be straining at the gnat and swallowing the camel. Spark plug noise is the least of your worries in that environment.

The second step is to generate the electrical noise in a controlled manner. This is usually more difficult than first imagined; one first imagines the simulation in terms of a waveform generator and a few spare parts, and then finds that a wideband power amplifier with a 200V dynamic range is also required. A good source of information on who supplies what noise-simulating equipment is the 1981 "ITEM" Directory and Design Guide (reference 6).

## Types of Failures and Failure Mechanisms

A major problem that EMI can cause in digital systems is intermittent operational malfunction. These software upsets occur when the system is in operation at the time an EMI source is activated, and are usually characterized by a loss of information or a jump in the execution of

the program to some random location in memory. The person who has to iron out such problems is tempted to say the program counter went crazy. There is usually no damage to the hardware, and normal operation can resume as soon as the EMI has passed or the source is de-activated. Resuming normal operation usually requires manual or automatic reset, and possibly re-entering of lost information.

Electrostatic discharges from operating personnel can cause not only software upsets, but also permanent ("hard") damage to the system. For this to happen the system doesn't even have to be in operation. Sometimes the permanent damage is latent, meaning the initial damage may be marginal and require further aggravation through operating stress and time before permanent failure takes place. Sometimes too the damage is hidden.

One ESD-related failure mechanism that has been identified has to do with the bias voltage on the substrate of the chip. On some CPU chips the substrate is held at -2.5V by a phase-shift oscillator working into a capacitor/diode clamping circuit. This is called a "charge pump" in chip-design circles. If the substrate wanders too far in either direction, program read errors are noted. Some designs have been known to allow electrostatic discharge currents to flow directly into port pins of an 8048. The resulting damage to the oxide causes an increase in leakage current, which loads down the charge pump, reducing the substrate voltage to a marginal or unacceptable level. The system is then unreliable or completely inoperative until the CPU chip is replaced. But if the CPU chip was subjected to a discharge spark once, it will eventually happen again.

Chips that have a grounded substrate, such as the 8748, can sometimes sustain some oxide damage without actually becoming inoperative. In this case the damage is present, and the increased leakage current is noted; however, since the substrate voltage retains its design value, the damage is largely hidden.

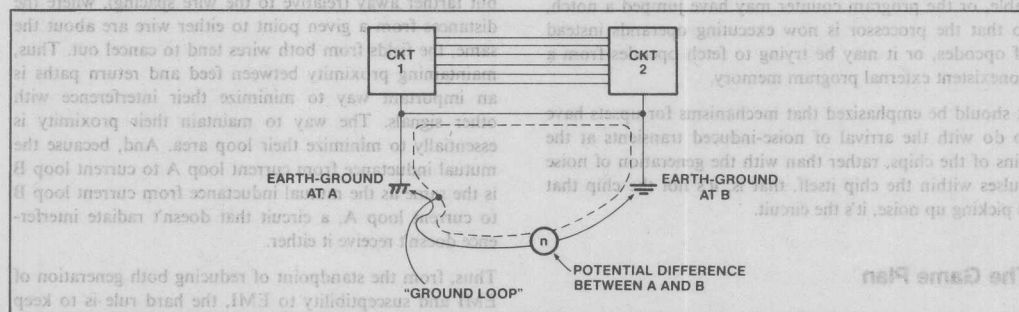


Figure 3. What a Ground Loop Is

It must therefore be recognized that connecting port pins unprotected to a keyboard or to anything else that is subject to electrostatic discharges, makes an extremely dangerous configuration. It doesn't make any difference what CPU chip is being used, or who makes it. If it connects unprotected to a keyboard, it will eventually be destroyed. Designing for an ESD-environment will be discussed further on.

We might note here that MOS chips are not the only components that are susceptible to permanent ESD damage. Bipolar and linear chips can also be damaged in this way. PN junctions are subject to a hard failure mechanism called thermal secondary breakdown, in which a current spike, such as from an electrostatic discharge, causes microscopically localized spots in the junction to approach melt temperatures. Low power TTL chips are subject to this type of damage, as are op-amps. Op-amps, in addition, often carry on-chip MOS capacitors which are directly across an external pin combination, and these are susceptible to dielectric breakdown.

We return now to the subject of software upsets. Noise transients can upset the chip through any pin, even an output pin, because every pin on the chip connects to the substrate through a pn junction. However, the most vulnerable pin is probably the VCC line, since it has direct access to all parts of the chip: every register, gate, flip-flop and buffer.

The menu of possible upset mechanisms is quite lengthy. A transient on the substrate at the wrong time will generally cause a program read error. A false level at a control input can cause an extraneous or misdirected opcode fetch. A disturbance on the supply line can flip a bit in the program counter or instruction register. A short interruption or reversal of polarity on the supply line can actually turn the processor off, but not long enough for the power-up reset capacitor to discharge. Thus when the transient ends, the chip starts up again without a reset.

A common failure mode is for the processor to lock itself into a tight loop. Here it may be executing the data in a table, or the program counter may have jumped a notch, so that the processor is now executing operands instead of opcodes, or it may be trying to fetch opcodes from a nonexistent external program memory.

It should be emphasized that mechanisms for upsets have to do with the arrival of noise-induced transients at the pins of the chips, rather than with the generation of noise pulses within the chip itself, that is, it's not the chip that is picking up noise, it's the circuit.

## The Game Plan

Prevention is usually cheaper than suppression, so first we'll consider some preventive methods that might help to

minimize the generation of noise voltages in the circuit. These methods involve grounding, shielding, and wiring techniques that are directed toward the mechanisms by which noise voltages are generated in the circuit. We'll also discuss methods of decoupling. Then we'll look at some schemes for making a graceful recovery from upsets that occur in spite of preventive measures. Lastly, we'll take another look at two special problem areas: electrostatic discharges and the automotive environment.

## Current Loops

The first thing most people learn about electricity is that current won't flow unless it can flow in a closed loop. This simple fact is sometimes temporarily forgotten by the overworked engineer who has spent the past several years mastering the intricacies of the DO loop, the timing loop, the feedback loop, and maybe even the ground loop. The simple current loop probably owes its apparent demise to the invention of the ground symbol. By a stroke of the pen one avoids having to draw the return paths of most of the current loops in the circuit. Then "ground" turns into an infinite current sink, so that any current that flows into it is gone and forgotten. Forgotten it may be, but it's not gone. It must return to its source, so that its path will by all the laws of nature form a closed loop.

The physical geometry of a given current loop is the key to why it generates EMI, why it's susceptible to EMI, and how to shield it. Specifically, it's the area of the loop that matters.

Any flow of current generates a magnetic field whose intensity varies inversely to the distance from the wire that carries the current. Two parallel wires conducting currents  $+I$  and  $-I$  (as in signal feed and return lines) would generate a nonzero magnetic field near the wires, where the distance from a given point to one wire is noticeably different from the distance to the other wire, but farther away (relative to the wire spacing), where the distances from a given point to either wire are about the same, the fields from both wires tend to cancel out. Thus, maintaining proximity between feed and return paths is an important way to minimize their interference with other signals. The way to maintain their proximity is essentially to minimize their loop area. And, because the mutual inductance from current loop A to current loop B is the same as the mutual inductance from current loop B to current loop A, a circuit that doesn't radiate interference doesn't receive it either.

Thus, from the standpoint of reducing both generation of EMI and susceptibility to EMI, the hard rule is to keep loop areas small. To say that loop areas should be minimized is the same as saying the circuit inductance should

be minimized. Inductance is by definition the constant of proportionality between current and the magnetic field it produces:  $\phi = LI$ . Holding the feed and return wires close together so as to promote field cancellation can be described either as minimizing the loop area or as minimizing  $L$ . It's the same thing.

## Shielding

There are three basic kinds of shields: shielding against capacitive coupling, shielding against inductive coupling, and RF shielding. Capacitive coupling is electric field coupling, so shielding against it amounts to shielding against electric fields. As will be seen, this is relatively easy. Inductive coupling is magnetic field coupling, so shielding against it is shielding against magnetic fields. This is a little more difficult. Strangely enough, this type of shielding does not in general involve the use of magnetic materials. RF shielding, the classical "metallic barrier" against all sorts of electromagnetic fields, is what most people picture when they think about shielding. Its effectiveness depends partly on the selection of the shielding material, but mostly, as it turns out, on the treatment of its seams and the geometry of its openings.

### SHIELDING AGAINST CAPACITIVE COUPLING

Capacitive coupling involves the passage of interfering signals through mutual or stray capacitances that aren't shown on the circuit diagram, but which the experienced engineer knows are there. Capacitive coupling to one's body is what would cause an unstable oscillator to change its frequency when the person reaches his hand over the circuit, for example. More importantly, in a digital system it causes crosstalk in multi-wire cables.

The way to block capacitive coupling is to enclose the circuit or conductor you want to protect in a metal shield. That's called an electrostatic or Faraday shield. If coverage is 100%, the shield does not have to be grounded, but it usually is, to ensure that circuit-to-shield capacitances go to signal reference ground rather than act as feedback and crosstalk elements. Besides, from a mechanical point of view, grounding it is almost inevitable.

A grounded Faraday shield can be used to break capacitive coupling between a noisy circuit and a victim circuit, as shown in Figure 4. Figure 4A shows two circuits capacitively coupled through the stray capacitance between them. In Figure 4B the stray capacitance is intercepted by a grounded Faraday shield, so that interference currents are shunted to ground. For example, a grounded plane can be inserted between PCBs (printed circuit boards) to eliminate most of the capacitive coupling between them.

Another application of the Faraday shield is in the elec-

trostatically shielded transformer. Here, a conducting foil is laid between the primary and secondary coils so as to intercept the capacitive coupling between them. If a system is being upset by AC line transients, this type of transformer may provide the fix. To be effective in this application, the shield must be connected to the green-wire ground.

### SHIELDING AGAINST INDUCTIVE COUPLING

With inductive coupling, the physical mechanism involved is a magnetic flux density  $B$  from some external interference source that links with a current loop in the victim circuit, and generates a voltage in the loop in accordance with Lenz's law:  $v = -NA(dB/dt)$ , where in this case  $N = 1$  and  $A$  is the area of the current loop in the victim circuit.

There are two aspects to defending a circuit against inductive pickup. One aspect is to try to minimize the offensive fields at their source. This is done by minimizing the area of the current loop at the source so as to promote field cancellation, as described in the section on current loops. The other aspect is to minimize the inductive pickup in the victim circuit by minimizing the area of that current loop, since, from Lenz's law, the induced voltage is proportional to this area. So the two aspects really involve the same corrective action: minimize the areas of the current loops. In other words, minimizing the offensiveness of a circuit inherently minimizes its susceptibility.

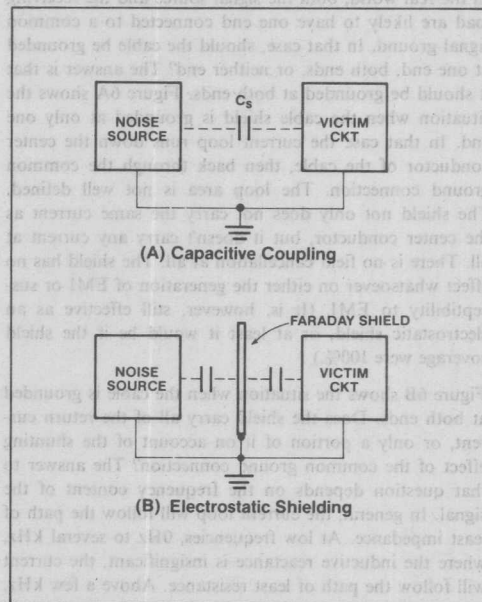


Figure 4. Use of Faraday Shield

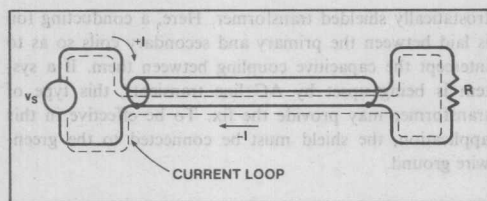


Figure 5. External to the Shield,  $\phi=0$

Shielding against inductive coupling means nothing more nor less than controlling the dimensions of the current loops in the circuit. We must look at four examples of this type of "shielding": the coaxial cable, the twisted pair, the ground plane, and the gridded-ground PCB layout.

**The Coaxial Cable** — Figure 5 shows a coaxial cable carrying a current  $I$  from a signal source to a receiving load. The shield carries the same current as the center conductor. Outside the shield, the magnetic field produced by  $+I$  flowing in the center conductor is cancelled by the field produced by  $-I$  flowing in the shield. To the extent that the cable is ideal in producing zero external magnetic field, it is immune to inductive pickup from external sources. The cable adds effectively zero area to the loop. This is true only if the shield carries the same current as the center conductor.

In the real world, both the signal source and the receiving load are likely to have one end connected to a common signal ground. In that case, should the cable be grounded at one end, both ends, or neither end? The answer is that it should be grounded at both ends. Figure 6A shows the situation when the cable shield is grounded at only one end. In that case the current loop runs down the center conductor of the cable, then back through the common ground connection. The loop area is not well defined. The shield not only does not carry the same current as the center conductor, but it doesn't carry any current at all. There is no field cancellation at all. The shield has no effect whatsoever on either the generation of EMI or susceptibility to EMI. (It is, however, still effective as an electrostatic shield, or at least it would be if the shield coverage were 100%.)

Figure 6B shows the situation when the cable is grounded at both ends. Does the shield carry all of the return current, or only a portion of it on account of the shunting effect of the common ground connection? The answer to that question depends on the frequency content of the signal. In general, the current loop will follow the path of least impedance. At low frequencies, 0Hz to several kHz, where the inductive reactance is insignificant, the current will follow the path of least resistance. Above a few kHz, where inductive reactance predominates, the current will follow the path of least inductance. The path of least

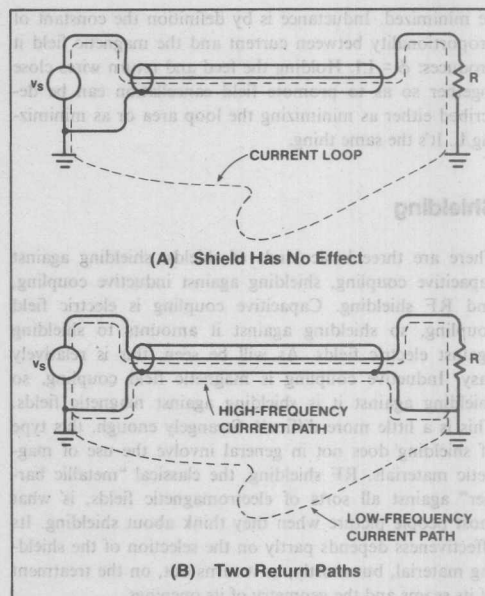


Figure 6. Use of Coaxial Cable

inductance is the path of minimum loop area. Hence, for higher frequencies the shield carries virtually the same current as the center conductor, and is therefore effective against both generation and reception of EMI.

Note that we have now introduced the famous "ground loop" problem, as shown in Figure 7A. Fortunately, a digital system has some built-in immunity to moderate ground loop noise. In a noisy environment, however, one can break the ground loop, and still maintain the shielding effectiveness of the coaxial cable, by inserting an optical coupler, as shown in Figure 7B. What the optical coupler does, basically, is allow us to re-define the signal source as being ungrounded, so that that end of the cable need not be grounded, and still lets the shield carry the same current as the center conductor. Obviously, if the signal source weren't grounded in the first place, the optical coupler wouldn't be needed.

**The Twisted Pair** — A cheaper way to minimize loop area is to run the feed and return wires right next to each other. This isn't as effective as a coaxial cable in minimizing loop area. An ideal coaxial cable adds zero area to the loop, whereas merely keeping the feed and return wires next to each other is bound to add a finite area.

However, two things work to make this cheaper method almost as good as a coaxial cable. First, real coaxial cables are not ideal. If the shield current isn't evenly distributed around the center conductor at every cross-



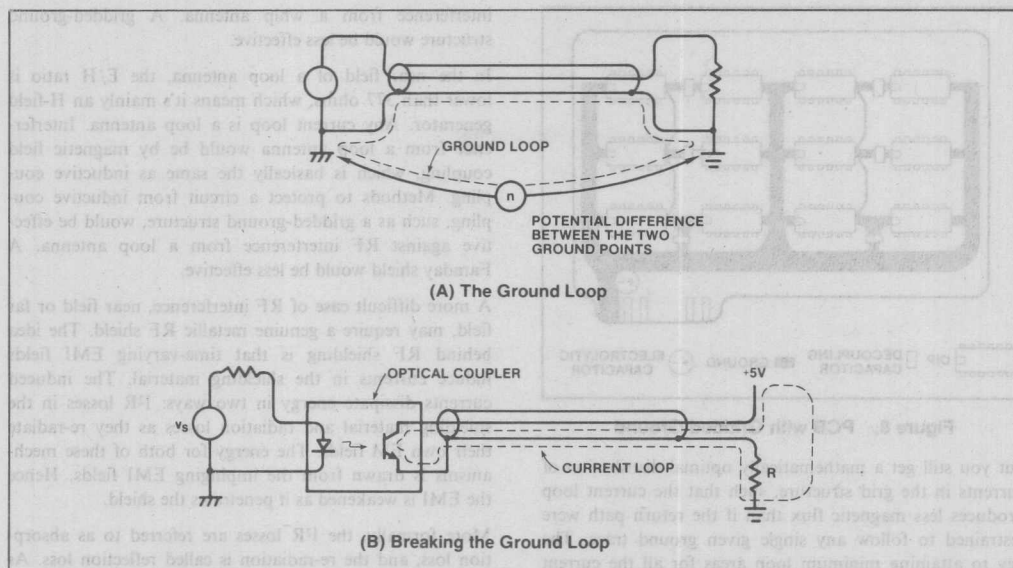


Figure 7. Use of Optical Coupler

section of the cable (it isn't), then field cancellation external to the shield is incomplete. If field cancellation is incomplete, then the effective area added to the loop by the cable isn't zero. Second, in the cheaper method the feed and return wires can be twisted together. This not only maintains their proximity, but the noise picked up in one twist tends to cancel out the noise picked up in the next twist down the line. Thus the "twisted pair" turns out to be about as good a shield against inductive coupling as coaxial cable is.

The twisted pair does not, however, provide electrostatic shielding (i.e., shielding against capacitive coupling). Another operational difference between them is that the coaxial cable works better at higher frequencies. This is primarily because the twisted pair adds more capacitive loading to the signal source than the coaxial cable does. The twisted pair is normally considered useful up to only about 1MHz, as opposed to near a GHz for the coaxial cable.

**The Ground Plane** — The best way to minimize loop areas when many current loops are involved is to use a ground plane. A ground plane is a conducting surface that is to serve as a return conductor for all the current loops in the circuit. Normally, it would be one or more layers of a multilayer PCB. All ground points in the circuit go not to a grounded trace on the PCB, but directly to the ground plane. This leaves each current loop in the circuit free to complete itself in whatever configuration yields minimum loop area (for frequencies wherein the

ground path impedance is primarily inductive).

Thus, if the feed path for a given signal zigzags its way across the PCB, the return path for this signal is free to zigzag right along beneath it on the ground plane, in such a configuration as to minimize the energy stored in the magnetic field produced by this current loop. Minimal magnetic flux means minimal effective loop area and minimal susceptibility to inductive coupling.

**The Gridded-Ground PCB Layout** — The next best thing to a ground plane is to lay out the ground traces on a PCB in the form of a grid structure, as shown in Figure 8. Laying horizontal traces on one side of the board and vertical traces on the other side allows the passage of signal and power traces. Wherever vertical and horizontal ground traces cross, they must be connected by a feed-through.

Have we not created here a network of "ground loops"? Yes, in the literal sense of the word, but loops in the ground layout on a PCB are not to be feared. Such inoffensive little loops have never caused as much noise pick-up as their avoidance has. Trying to avoid innocent little loops in the ground layout, PCB designers have forced current loops into geometries that could swallow a whale. That is exactly the wrong thing to do.

The gridded ground structure works almost as well as the ground plane, as far as minimizing loop area is concerned. For a given current loop, the primary return path may have to zig once in a while where its feed path zags,



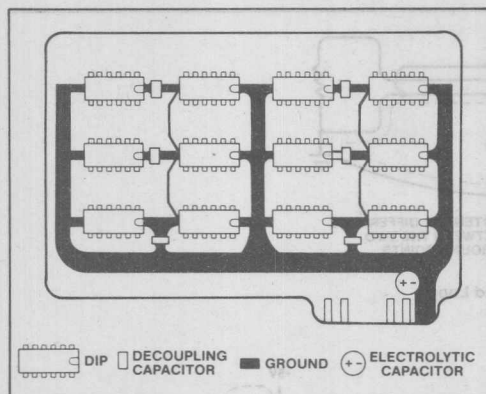


Figure 8. PCB with Gridded Ground

but you still get a mathematically optimal distribution of currents in the grid structure, such that the current loop produces less magnetic flux than if the return path were restrained to follow any single given ground trace. The key to attaining minimum loop areas for all the current loops together is to let the ground currents distribute themselves around the entire area of the board as freely as possible. They want to minimize their own magnetic field. Just let them.

#### RF SHIELDING

A time-varying electric field generates a time-varying magnetic field, and vice versa. Far from the source of a time-varying EM field, the ratio of the amplitudes of the electric and magnetic fields is always 377 ohms. Up close to the source of the fields, however, this ratio can be quite different, and dependent on the nature of the source. Where the ratio is near 377 ohms is called the far field, and where the ratio is significantly different from 377 ohms is called the near field. The ratio itself is called the wave impedance, E/H.

The near field goes out about 1/6 of a wavelength from the source. At 1MHz this is about 150 feet, and at 10MHz it's about 15 feet. That means if an EMI source is in the same room with the victim circuit, it's likely to be a near field problem. The reason this matters is that in the near field an RF interference problem could be almost entirely due to E-field coupling or H-field coupling, and that could influence the choice of an RF shield or whether an RF shield will help at all.

In the near field of a whip antenna, the E/H ratio is higher than 377 ohms, which means it's mainly an E-field generator. A wire-wrap post can be a whip antenna. Interference from a whip antenna would be by electric field coupling, which is basically capacitive coupling. Methods to protect a circuit from capacitive coupling, such as a Faraday shield, would be effective against RF

interference from a whip antenna. A gridded-ground structure would be less effective.

In the near field of a loop antenna, the E/H ratio is lower than 377 ohms, which means it's mainly an H-field generator. Any current loop is a loop antenna. Interference from a loop antenna would be by magnetic field coupling, which is basically the same as inductive coupling. Methods to protect a circuit from inductive coupling, such as a gridded-ground structure, would be effective against RF interference from a loop antenna. A Faraday shield would be less effective.

A more difficult case of RF interference, near field or far field, may require a genuine metallic RF shield. The idea behind RF shielding is that time-varying EMI fields induce currents in the shielding material. The induced currents dissipate energy in two ways: I<sup>2</sup>R losses in the shielding material and radiation losses as they re-radiate their own EM fields. The energy for both of these mechanisms is drawn from the impinging EMI fields. Hence the EMI is weakened as it penetrates the shield.

More formally, the I<sup>2</sup>R losses are referred to as absorption loss, and the re-radiation is called reflection loss. As it turns out, absorption loss is the primary shielding mechanism for H-fields, and reflection loss is the primary shielding mechanism for E-fields. Reflection loss, being a surface phenomenon, is pretty much independent of the thickness of the shielding material. Both loss mechanisms, however, are dependent on the frequency ( $\omega$ ) of the impinging EMI field, and on the permeability ( $\mu$ ) and conductivity ( $\sigma$ ) of the shielding material. These loss mechanisms vary approximately as follows:

$$\text{reflection loss to an E-field (in dB)} \sim \log \frac{\sigma}{\omega \mu}$$

$$\text{absorption loss to an H-field (in dB)} \sim t \sqrt{\omega \sigma \mu}$$

where  $t$  is the thickness of the shielding material.

The first expression indicates that E-field shielding is more effective if the shield material is highly conductive, and less effective if the shield is ferromagnetic, and that low-frequency fields are easier to block than high-frequency fields. This is shown in Figure 9.

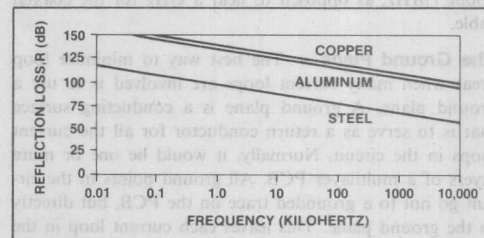


Figure 9. E-Field Shielding

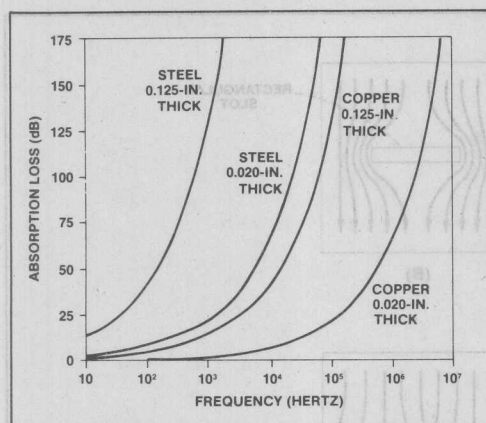


Figure 10. H-Field Shielding

Copper and aluminum both have the same permeability, but copper is slightly more conductive, and so provides slightly greater reflection loss to an E-field. Steel is less effective for two reasons. First, it has a somewhat elevated permeability due to its iron content, and, second, as tends to be the case with magnetic materials, it is less conductive.

On the other hand, according to the expression for absorption loss to an H-field, H-field shielding is more effective at higher frequencies and with shield material that has both high conductivity and high permeability. In practice, however, selecting steel for its high permeability involves some compromise in conductivity. But the increase in permeability more than makes up for the decrease in conductivity, as can be seen in Figure 10. This figure also shows the effect of shield thickness.

A composite of E-field and H-field shielding is shown in Figure 11. However, this type of data is meaningful only in the far field. In the near field the EMI could be 90% H-field, in which case the reflection loss is irrelevant. It would be advisable then to beef up the absorption loss, at the expense of reflection loss, by choosing steel. A better conductor than steel might be less expensive, but quite ineffective.

A different shielding mechanism that can be taken advantage of for low frequency magnetic fields is the ability of a high permeability material such as mumetal to divert the field by presenting a very low reluctance path to the magnetic flux. Above a few kHz, however, the permeability of such materials is the same as steel.

In actual fact the selection of a shielding material turns out to be less important than the presence of seams, joints and holes in the physical structure of the enclosure. The shielding mechanisms are related to the induction of currents in the shield material, but the currents must be

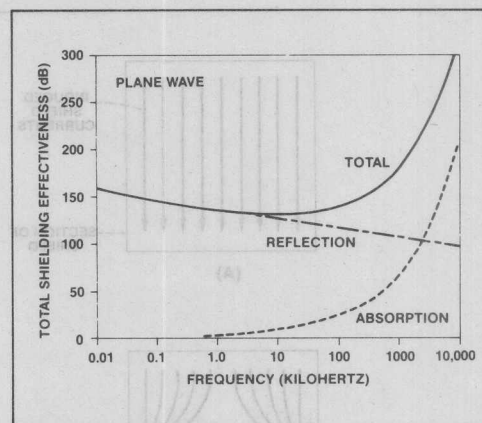


Figure 11. E- and H-Field Shielding

allowed to flow freely. If they have to detour around slots and holes, as shown in Figure 12, the shield loses much of its effectiveness.

As can be seen in Figure 12, the severity of the detour has less to do with the area of the hole than it does with the geometry of the hole. Comparing Figure 12C with 12D shows that a long narrow discontinuity such as a seam can cause more RF leakage than a line of holes with larger total area. A person who is responsible for designing or selecting rack or chassis enclosures for an EMI environment needs to be familiar with the techniques that are available for maintaining electrical continuity across seams. Information on these techniques is available in the references.

## Grounds

There are two kinds of grounds: earth-ground and signal ground. The earth is not an equipotential surface, so earth ground potential varies. That and its other electrical properties are not conducive to its use as a return conductor in a circuit. However, circuits are often connected to earth ground for protection against shock hazards. The other kind of ground, signal ground, is an arbitrarily selected reference node in a circuit—the node with respect to which other node voltages in the circuit are measured.

## SAFETY GROUND

The standard 3-wire single-phase AC power distribution system is represented in Figure 13. The white wire is earth-grounded at the service entrance. If a load circuit has a metal enclosure or chassis, and if the black wire develops a short to the enclosure, there will be a shock hazard to operating personnel, unless the enclosure itself is earth-grounded. If the enclosure is earth-grounded, a

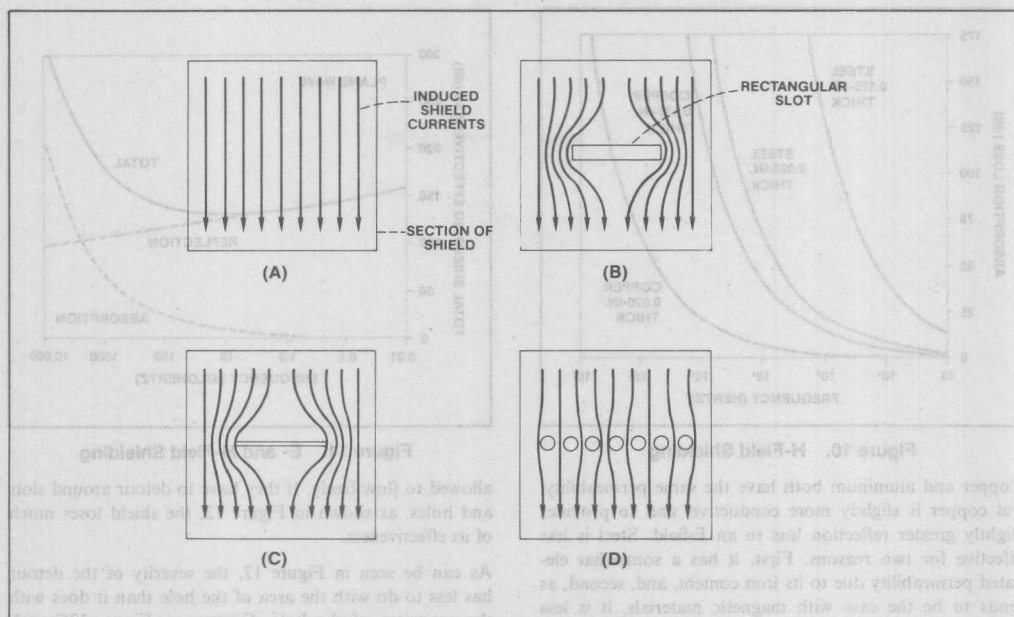


Figure 12. Effect of Shield Discontinuity on Magnetically Induced Shield Current

short results in a blown fuse rather than a "hot" enclosure. The earth-ground connection to the enclosure is called a safety ground. The advantage of the 3-wire power system is that it distributes a safety ground along with the power.

Note that the safety-ground wire carries no current, except in case of a fault, so that at least for low frequencies it's at earth-ground potential along its entire length. The white wire, on the other hand, may be several volts off ground, due to the IR drop along its length.

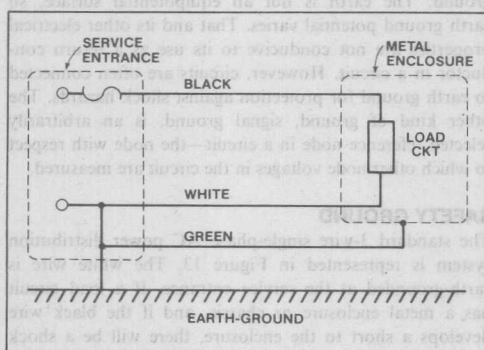


Figure 13. Single-Phase Power Distribution

### SIGNAL GROUND

Signal ground is a single point in a circuit that is designated to be the reference node for the circuit. Commonly, wires that connect to this single point are also referred to as "signal ground." In some circles "power supply common" or PSC is the preferred terminology for these conductors. In any case, the manner in which these wires connect to the actual reference point is the basis of distinction among three kinds of signal-ground wiring methods: series, parallel, and multipoint. These methods are shown in Figure 14.

The series connection is pretty common because it's simple and economical. It's the noisiest of the three, however, due to common ground impedance coupling between the circuits. When several circuits share a ground wire, currents from one circuit, flowing through the finite impedance of the common ground line, cause variations in the ground potential of the other circuits. Given that the currents in a digital system tend to be spiked, and that the common impedance is mainly inductive reactance, the variations could be bad enough to cause bit errors in high current or particularly noisy situations.

The parallel connection eliminates common ground impedance problems, but uses a lot of wire. Other disadvantages are that the impedance of the individual ground lines can be very high, and the ground lines themselves can become sources of EMI.

In the multipoint system, ground impedance is minimized by using a ground plane with the various circuits connected to it by very short ground leads. This type of connection would be used mainly in RF circuits above 10MHz.

### PRACTICAL GROUNDING

A combination of series and parallel ground-wiring methods can be used to trade off economic and the various electrical considerations. The idea is to run series connections for circuits that have similar noise properties, and connect them at a single reference point, as in the parallel method, as shown in Figure 15.

In Figure 15, "noisy signal ground" connects to things like motors and relays. Hardware ground is the safety ground connection to chassis, racks, and cabinets. It's a mistake to use the hardware ground as a return path for signal currents because it's fairly noisy (for example, it's the hardware ground that receives an ESD spark) and tends to have high resistance due to joints and seams.

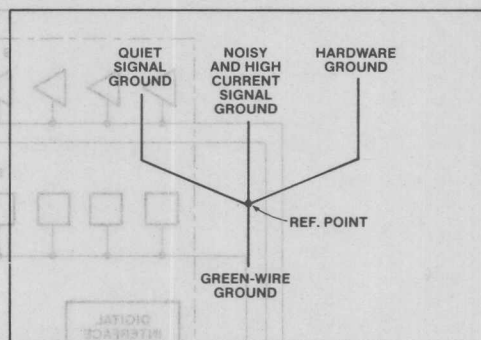


Figure 15. Parallel Connection of Series Grounds

Screws and bolts don't always make good electrical connections because of galvanic action, corrosion, and dirt. These kinds of connections may work well at first, and then cause mysterious maladies as the system ages.

Figure 16 illustrates a grounding system for a 9-track digital tape recorder, showing an application of the series/parallel ground-wiring method.

Figure 17 shows a similar separation of grounds at the PCB level. Currents in multiplexed LED displays tend to put a lot of noise on the ground and supply lines because of the constant switching and changing involved in the scanning process. The segment driver ground is relatively quiet, since it doesn't conduct the LED currents. The digit driver ground is noisier, and should be provided with a separate path to the PCB ground terminal, even if the PCB ground layout is gridded. The LED feed and return current paths should be laid out on opposite sides of the board like parallel flat conductors.

Figure 18 shows right and wrong ways to make ground connections in racks. Note that the safety ground connections from panel to rack are made through ground straps, not panel screws. Rack 1 correctly connects signal ground to rack ground only at the single reference point. Rack 2 incorrectly connects signal ground to rack ground at two points, creating a ground loop around points 1, 2, 3, 4, 1.

Breaking the "electronics ground" connection to point 1 eliminates the ground loop, but leaves signal ground in rack 2 sharing a ground impedance with the relatively noisy hardware ground to the reference point; in fact, it may end up using hardware ground as a return path for signal and power supply currents. This will probably cause more problems than the ground loop.

### BRAIDED CABLE

Ground impedance problems can be virtually eliminated by using braided cable. The reduction in impedance is due to skin effect: At higher frequencies the current tends to flow along the surface of a conductor rather than uni-

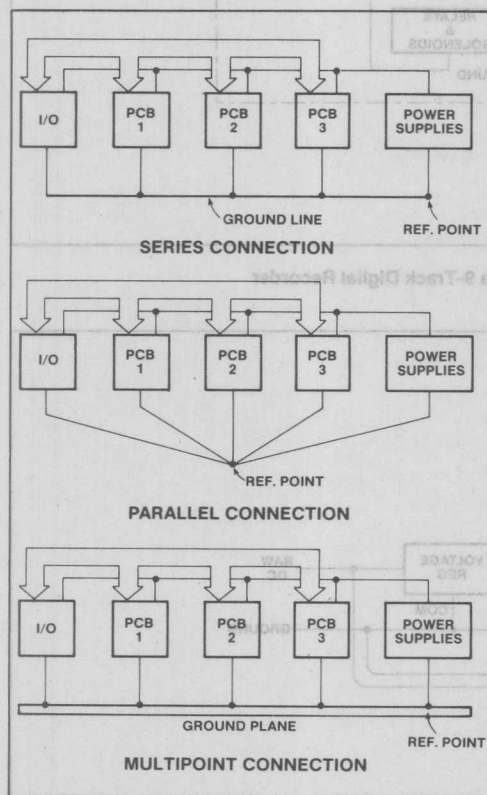


Figure 14. Three Ways to Wire the Grounds



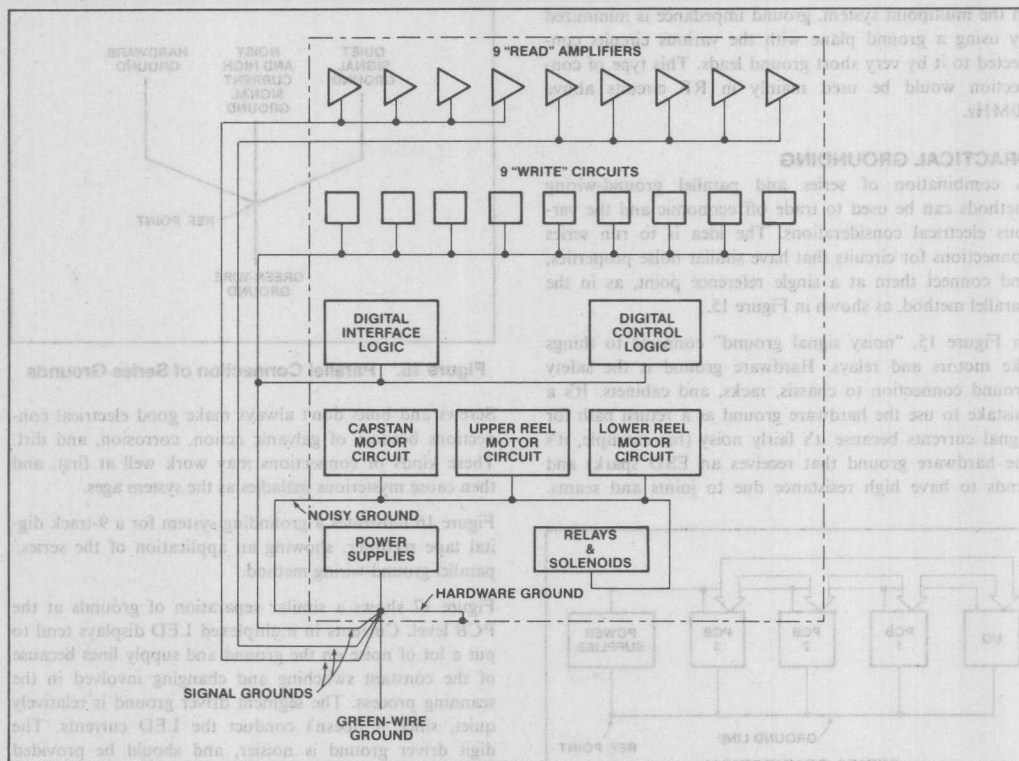


Figure 16. Ground System in a 9-Track Digital Recorder

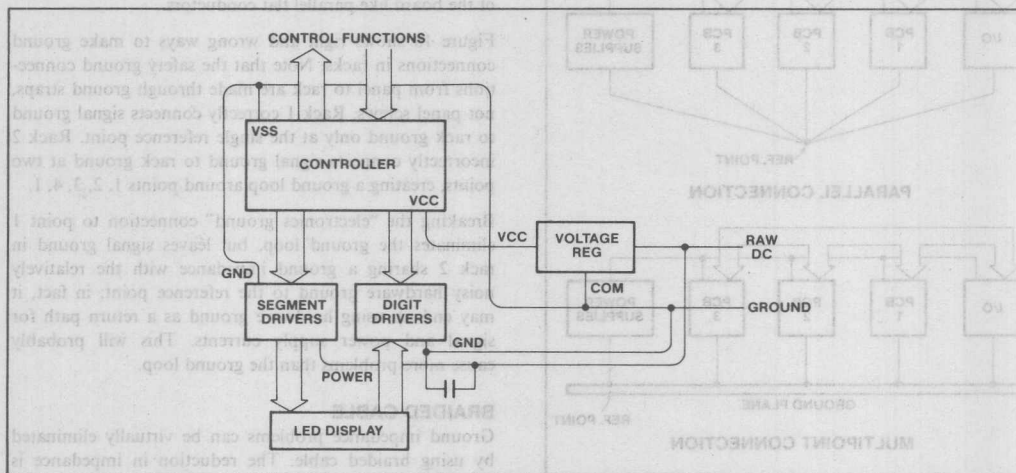
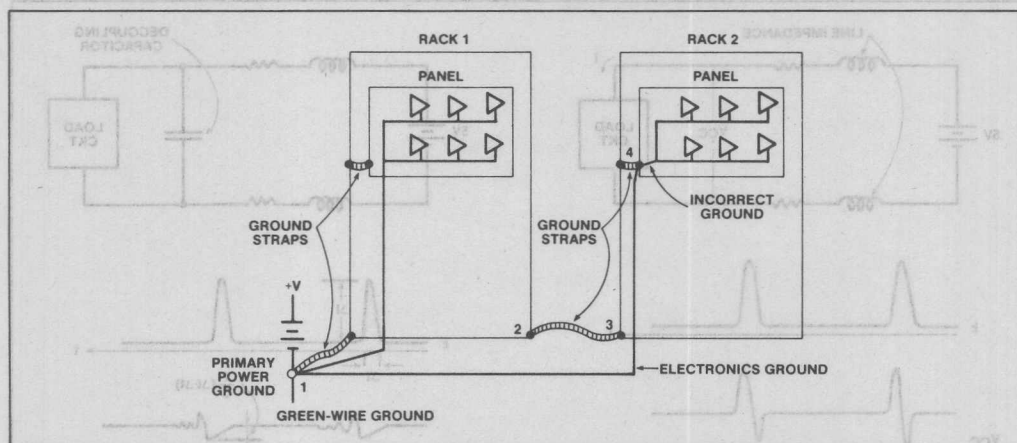


Figure 17. Separate Ground for Multiplexed LED Display





**Figure 18. Electronic Circuits Mounted in Equipment Racks Should Have Separate Ground Connections. Rack 1 Shows Correct Grounding, Rack 2 Shows Incorrect Grounding**

formly through its bulk. While this effect tends to increase the impedance of a given conductor, it also indicates the way to minimize impedance, and that is to manipulate the shape of the cross-section so as to provide more surface area. For its bulk, braided cable is almost pure surface.

### Power Supply Distribution and Decoupling

The main consideration for power supply distribution lines is, as for signal lines, to minimize the areas of the current loops. But the power supply lines take on an importance that no signal line has when one considers the fact that these lines have access to every PC board in the system. The very extensiveness of the supply current loops makes it difficult to keep loop areas small. And, a noise glitch on a supply line is a glitch delivered to every board in the system.

The power supply provides low-frequency current to the load, but the inductance of the board-to-board and chip-to-chip distribution network makes it difficult for the power supply to maintain VCC specs on the chip while providing the current spikes that a digital system requires. In addition, the power supply current loop is a very large one, which means there will be a lot of noise pick-up. Figure 19A shows a load circuit trying to draw current spikes from a supply voltage through the line impedance. To the VCC waveform shown in that figure should be added the inductive pick-up associated with a large loop area.

Adding a decoupling capacitor solves two problems: The capacitor acts as a nearby source of charge to supply the current spikes through a smaller line impedance, and it

defines a much smaller loop area for the higher frequency components of EMI. This is illustrated in Figure 19B, which shows the capacitor supplying the current spike, during which VCC drops from 5V by the amount indicated in the figure. Between current spikes the capacitor recovers through the line impedance.

One should resist the temptation to add a resistor or an inductor to the decoupler so as to form a genuine RC or LC low-pass filter because that slows down the speed with which the decoupler cap can be refreshed. Good filtering and good decoupling are not necessarily the same thing.

The current loop for the higher frequency currents, then, is defined by the decoupling cap and the load circuit, rather than by the power supply and the load circuit. For the decoupling cap to be able to provide the current spikes required by the load, the inductance of this current loop must be kept small, which is the same as saying the loop area must be kept small. This is also the requirement for minimizing inductive pick-up in the loop.

There are two kinds of decoupling caps: board decouplers and chip decouplers. A board decoupler will normally be a 10 to 100 $\mu$ f electrolytic capacitor placed near to where the power supply enters the PC board, but its placement is relatively non-critical. The purpose of the board decoupler is to refresh the charge on the chip decouplers. The chip decouplers are what actually provide the current spikes to the chips. A chip decoupler will normally be a 0.1 to 1 $\mu$ f ceramic capacitor placed near the chip and connected to the chip by traces that minimize the area of the loop formed by the cap and the chip. If a chip decoupler is not properly placed on the board, it will be ineffective as a decoupler and will serve only to increase

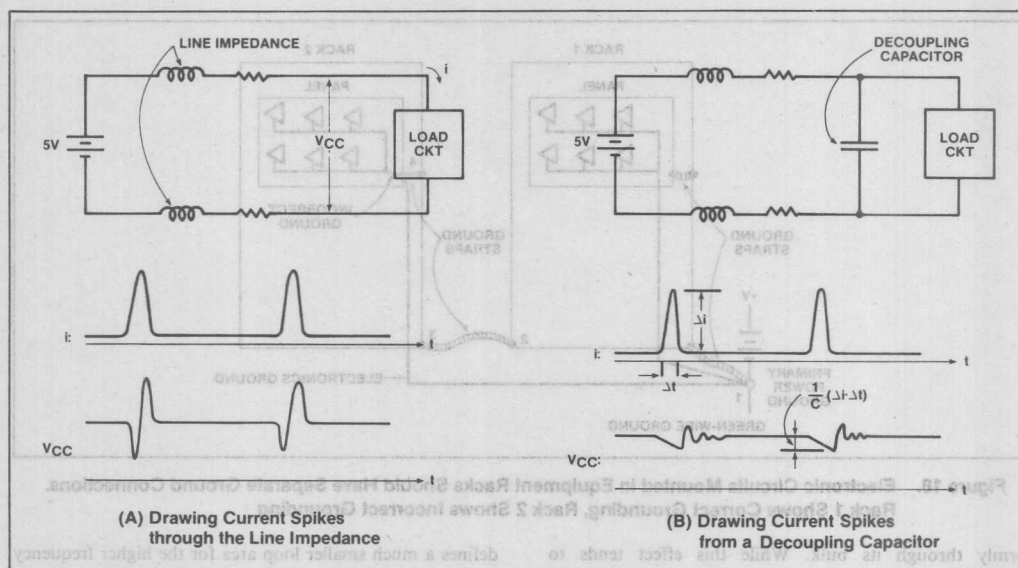


Figure 19. What a Decoupling Capacitor Does

the cost of the board. Good and bad placement of decoupling capacitors are illustrated in Figure 20.

Power distribution traces on the PC board need to be laid out so as to obtain minimal area (minimal inductance) in the loops formed by each chip and its decoupler, and by the chip decouplers and the board decoupler. One way to accomplish this goal is to use a power plane. A power plane is the same as a ground plane, but at  $V_{CC}$  potential. More economically, a power grid similar to the ground grid previously discussed (Figure 8) can be used. Actually, if the chip decoupling loops are small, other aspects of the power layout are less critical. In other words, power planes and power gridding aren't needed, but power traces *should* be laid in the closest possible proximity to ground traces, preferably so that

each power trace is on the direct opposite side of the board from a ground trace.

Special-purpose power supply distribution buses which mount on the PCB are available. The buses use a parallel flat conductor configuration, one conductor being a  $V_{CC}$  line and the other a ground line. Used in conjunction with a gridded ground layout, they not only provide a low-inductance distribution system, but can themselves form part of the ground grid, thus facilitating the PCB layout. The buses are available with and without enhanced bus capacitance, under the names Mini/Bus® and Q/PAC® from Rogers Corp. (5750 E. McKellips, Mesa, AZ 85205).

## SELECTING THE VALUE OF THE DECOUPLING CAP

The effectiveness of the decoupling cap has a lot to do with the way the power and ground traces connect this capacitor to the chip. In fact, the area formed by this loop is more important than the value of the capacitance. Then, given that the area of this loop is indeed minimal, it can generally be said that the larger the value of the decoupling cap, the more effective it is, if the cap has a mica, ceramic, glass, or polystyrene dielectric.

It's often said, and not altogether accurately, that the chip decoupler shouldn't have too large a value. There are two reasons for this statement. One is that some capacitors, because of the nature of their dielectrics, tend to become inductive or lossy at higher frequencies. This is true of

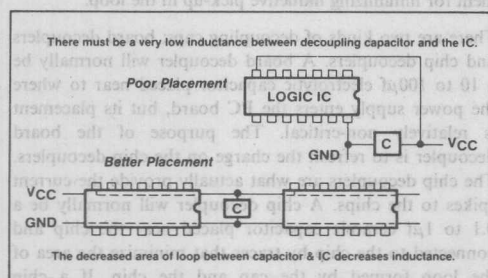


Figure 20. Placement of Decoupling Capacitors

electrolytic capacitors, but mica, glass, ceramic, and polystyrene dielectrics work well to several hundred MHz. The other reason cited for not using too large a capacitance has to do with lead inductance.

The capacitor with its lead inductance forms a series LC circuit. Below the frequency of series resonance, the net impedance of the combination is capacitive. Above that frequency, the net impedance is inductive. Thus a decoupling capacitor is capacitive only below the frequency of series resonance. This frequency is given by

$$f_0 = \frac{1}{2\pi\sqrt{LC}}$$

where C is the decoupling capacitance and L is the lead inductance between the capacitor and the chip. On a PC board this inductance is determined by the layout, and is the same whether the capacitor dropped into the PCB holes is 0.001 $\mu$ f or 1 $\mu$ f. Thus, increasing the capacitance lowers the series resonant frequency. In fact, according to the resonant frequency formula, increasing C by a factor of 100 lowers the resonant frequency by a factor of 10.

Figures quoted on the series resonant frequency of a 0.01 $\mu$ f capacitor run from 10 to 15MHz, depending on the lead length. If these numbers were accurate, a 1 $\mu$ f capacitor in the same position on the board would have a resonant frequency of 1.0 to 1.5MHz, and as a decoupler would do more harm than good. However, the numbers are based on a presumed inductance of a given length of wire (the lead length). It should be noted that a "length of wire" has no inductance at all, strictly speaking. Only a complete current loop has inductance, and the inductance depends on the geometry of the loop. Figures quoted on the inductance of a length of wire are based on a presumably "very large" loop area, such that the magnetic field produced by the return current has no cancellation effect on the field produced by the current in the given length of wire. Such a loop geometry is not and should not be the case with the decoupling loop.

Figure 21 shows VCC waveforms, measured between pins 40 and 20 (VCC and VSS) of an 8751 CPU, for several conditions of decoupling on a PC board that has a decoupling loop area slightly larger than necessary. These photographs show the effects of increasing the decoupling capacitance and decreasing the area of the decoupling loop. The indications are that a 1 $\mu$ f capacitor is better than a 0.1 $\mu$ f capacitor, which in turn is better than nothing, and that the board should have been laid out with more attention paid to the area of the decoupling loop.

Figure 21E was obtained using a special-purpose experimental capacitor designed by Rogers Corp. (Q-Pac Division, Mesa, AZ) for use as a decoupler. It consists of two parallel plates, the length of a 40-pin DIP, separated by a

ceramic dielectric. Sandwiched between the CPU chip and the PCB (or between the CPU socket and the PCB), it makes connection to pins 40 and 20, forming a leadless decoupling capacitor. It is obviously a configuration of minimal inductance. Unfortunately, the particular sample tested had only 0.07 $\mu$ f of capacitance and so was unable to prevent the 1MHz ripple as effectively as the configuration of Figure 21D. It seems apparent, though, that with more capacitance this part will alleviate a lot of decoupling problems.

## THE CASE FOR ON-BOARD VOLTAGE REGULATION

To complicate matters, supply line glitches aren't always picked up in the distribution networks, but can come from the power supply circuit itself. In that case, a well-designed distribution network faithfully delivers the glitch throughout the system. The VCC glitch in Figure 22 was found to be coming from within a bench power supply in response to the EMP produced by an induction coil spark generator that was being used at Intel during a study of noise sensitivity. The VCC glitch is about 400mV high and some 20 $\mu$ sec in duration. Normal board decoupling techniques were ineffective in removing it, but adding an on-board voltage regulator chip did the job.

Thus, a good case can be made in favor of using a voltage regulator chip on each PCB, instead of doing all the voltage regulation at the supply circuit. This eases requirements on the heat-sinking at the supply circuit, and alleviates much of the distribution and board decoupling headaches. However, it also brings in the possibility that different boards would be operating at slightly different VCC levels due to tolerance in the regulator chips; this then leads to slightly different logic levels from board to board. The implications of that may vary from nothing to latch-up, depending on what kinds of chips are on the boards, and how they react to an input "high" that is perhaps 0.4V higher than local VCC.

## Recovering Gracefully from a Software Upset

Even when one follows all the best guidelines for designing for a noisy environment, it's always possible for a noise transient to occur which exceeds the circuit's immunity level. In that case, one can strive at least for a graceful recovery.

Graceful recovery schemes involve additional hardware and/or software which is supposed to return the system to a normal operating mode after a software upset has occurred. Two decisions have to be made: How to recognize when an upset has occurred, and what to do about it.

If the designer knows what kinds and combinations of

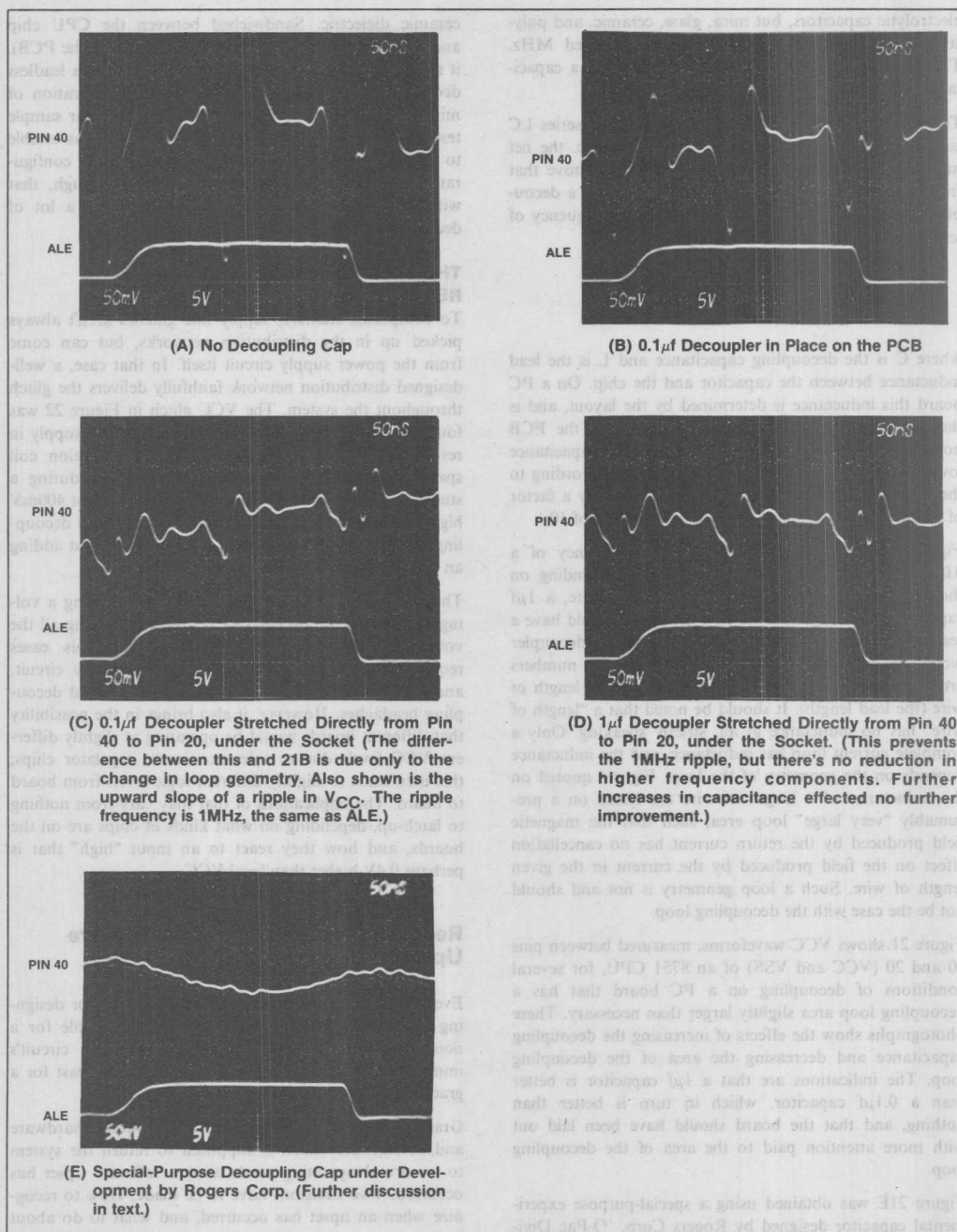


Figure 21. Noise on  $V_{CC}$  Line



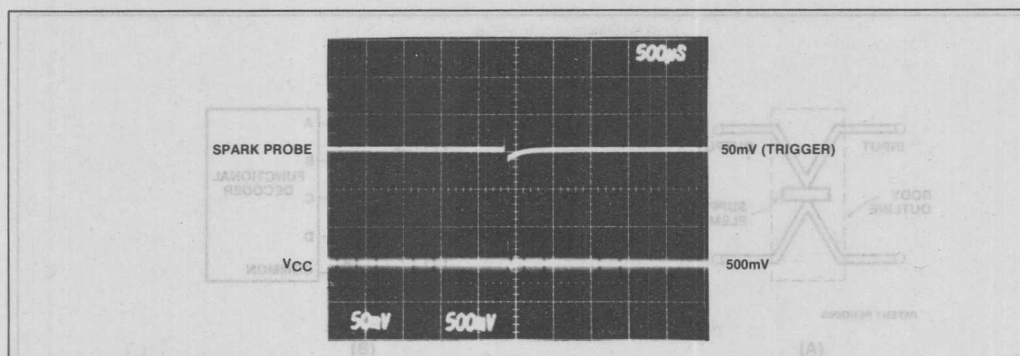


Figure 22. EMP-Induced Glitch

outputs can legally be generated by the system, he can use gates to recognize and flag the occurrence of an illegal state of affairs. The flag can then trigger a jump to a recovery routine which then may check or re-initialize data, perhaps output an error message, or generate a simple reset.

The most reliable scheme is to use a so-called watchdog circuit. Here the CPU is programmed to generate a periodic signal as long as the system is executing instructions in an expected manner. The periodic signal is then used to hold off a circuit that will trigger a jump to a recovery routine. The periodic signal needs to be AC-coupled to the trigger circuit so that a "stuck-at" fault won't continue to hold off the trigger. Then, if the processor locks up someplace, the periodic signal is lost and the watchdog triggers a reset.

In practice, it may be convenient to drive the watchdog circuit with a signal which is being generated anyway by the system. One needs to be careful, however, that an upset does in fact discontinue that signal. Specifically, for example, one could use one of the digit drive signals going to a multiplexed display. But display scanning is often handled in response to a timer-interrupt, which may continue operating even though the main program is in a failure mode. Even so, with a little extra software, the signal can be used to control the watchdog (see reference 8 on this).

Simpler schemes can work well for simpler systems. For example, if a CPU isn't doing anything but scanning and decoding a keyboard, there's little to lose and much to gain by simply resetting it periodically with an astable multivibrator. It only takes about  $13\mu\text{sec}$  (at 6MHz) to reset an 8048 if the clock oscillator is already running.

A zero-cost measure is simply to fill all unused program memory with NOPs and JMPs to a recovery routine. The effectiveness of this method is increased by writing the program in segments that are separated by NOPs and

JMPs. It's still possible, of course, to get hung up in a data table or something. But you get a lot of protection, for the cost.

Further discussion of graceful recovery schemes can be found in reference 13.

## Special Problem Areas

### ESD

MOS chips have some built-in protection against a static charge build-up on the pins, as would occur during normal handling, but there's no protection against the kinds of current levels and rise times that occur in a genuine electrostatic spark. These kinds of discharges can blow a crater in the silicon.

It must be recognized that connecting CPU pins unprotected to a keyboard or to anything else that is subject to electrostatic discharges makes an extremely fragile configuration. Buffering them is the very least one can do. But buffering doesn't completely solve the problem, because then the buffer chips will sustain the damage (even TTL); therefore, one might consider mounting the buffer chips in sockets for ease of replacement.

Transient suppressors, such as the TranZorbs® made by General Semiconductor Industries (Tempe, AZ), may in the long run provide the cheapest protection if their "zero inductance" structure is used. The structure and circuit application are shown in Figure 23.

The suppressor element is a pn junction that operates like a Zener diode. Back-to-back units are available for AC operation. The element is more or less an open circuit at normal system voltage (the standoff voltage rating for the device), and conducts like a Zener diode at the clamping voltage.

The lead inductance in the conventional transient suppressor package makes the conventional package essen-



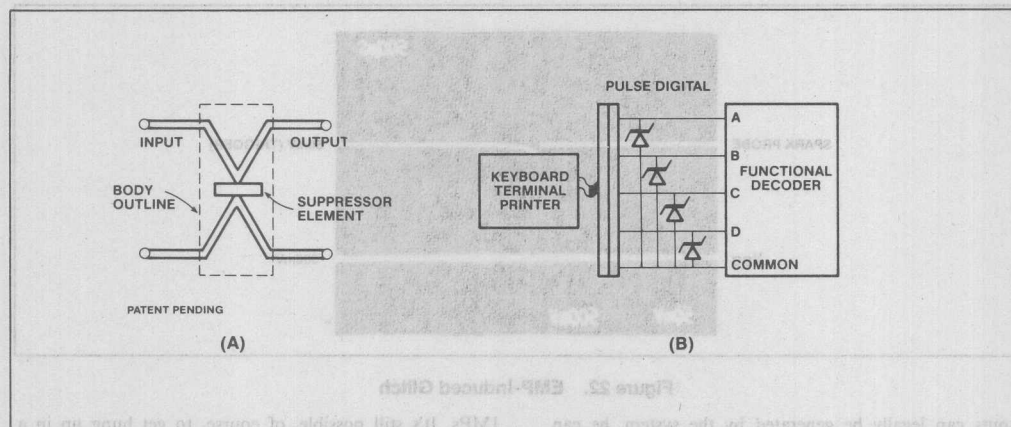


Figure 23. "Zero-inductance" Structure and Use in Circuit

tially useless for protection against ESD pulses, owing to the fast rise of these pulses. The "zero inductance" units are available singly in a 4-pin DIP, and in arrays of four to a 16-pin DIP for PCB level protection. In that application they should be mounted in close proximity to the chips they protect.

In addition, metal enclosures or frames or parts that can receive an ESD spark should be connected by braided cable to the green-wire ground. Because of the ground impedance, ESD current shouldn't be allowed to flow through any signal ground, even if the chips are protected by transient suppressors. A 35kV ESD spark can always spare a few hundred volts to drive a fast current pulse down a signal ground line if it can't find a braided cable to follow. Think how delighted your 8048 will be to find its VSS pin about 250V higher than VCC for a few 10s of nanoseconds.

#### THE AUTOMOTIVE ENVIRONMENT

The automobile presents an extremely hostile environment for electronic systems. There are several parts to it:

1. Temperature extremes from  $-40^{\circ}\text{C}$  to  $+125^{\circ}\text{C}$  (under the hood) or  $+85^{\circ}\text{C}$  (in the passenger compartment)
2. Electromagnetic pulses from the ignition system
3. Supply line transients that will knock your socks off

One needs to take a long, careful look at the temperature extremes. The allowable storage temperature range for most Intel MOS chips is  $-65^{\circ}\text{C}$  to  $+150^{\circ}\text{C}$ , although some chips have a maximum storage temperature rating of  $+125^{\circ}\text{C}$ . In operation (or "under bias," as the data sheets say) the allowable ambient temperature range depends on the product grade, as follows:

| Grade      | Ambient Temperature |        |
|------------|---------------------|--------|
|            | min.                | max.   |
| Commercial | 0                   | 70     |
| Industrial | $-40$               | $+85$  |
| Automotive | $-40$               | $+110$ |
| Military   | $-55$               | $+125$ |

The different product grades are actually the same chip, but tested according to different standards. Thus, a given commercial-grade chip might actually pass military temperature requirements, but not have been tested for it. (Of course, there are other differences in grading requirements having to do with packaging, burn-in, traceability, etc.)

In any case, it's apparent that commercial-grade chips can't be used safely in automotive applications, not even in the passenger compartment. Industrial-grade chips can be used in the passenger compartment, and automotive or military chips are required in under-the-hood applications.

Ignition noise, CB radios, and that sort of thing are probably the least of your worries. In a poorly designed system, or in one that has not been adequately tested for the automotive environment, this type of EMI might cause a few software upsets, but not destroy chips.

The major problem, and the one that seems to come as the biggest surprise to most people, is the line transients. Regrettably, the 12V battery is not actually the source of power when the car is running. The charging system is, and it's not very clean. The only time the battery is the real source of power is when the car is first being started, and in that condition the battery terminals may be delivering about 5 or 6V. Below is a brief description of the major idiosyncracies of the "12V" automotive power line.

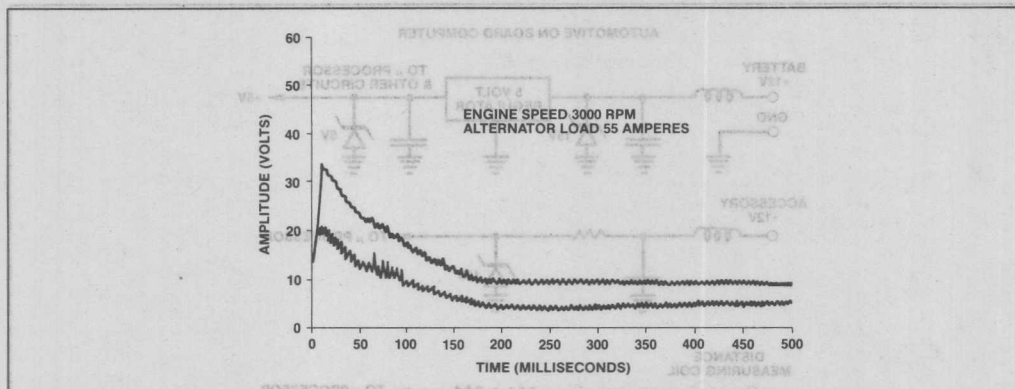


Figure 24. Typical Load Dump Transients

- An abrupt reduction in the alternator load causes a positive voltage transient called "load dump." In a load dump transient the line voltage rises to 20 or 30V in a few msec, then decays exponentially with a time constant of about 100msec, as shown in Figure 24. Much higher peak voltages and longer decay times have also been reported. The worst case load dump is caused by disconnecting a low battery from the alternator circuit while the alternator is running. Normally this would happen intermittently when the battery terminal connections are defective.
- When the ignition is turned off, as the field excitation decays, the line voltage can go to between -40 and -100V for 100 msec or more.
- Miscellaneous solenoid switching transients, such as the one shown in Figure 25, can drive the line to + or -200 to 400V for several  $\mu$ sec.
- Mutual coupling between unshielded wires in long harnesses can induce 100 and 200V transients in unprotected circuits.

What all this adds up to is that people in the business of building systems for automotive applications need a comprehensive testing program. An SAE guideline which describes the automotive environment is available to designers: SAE J1211, "Recommended Environmental Practices for Electronic Equipment Design," 1980 SAE Handbook, Part 1, pp. 22.80-22.96.

Some suggestions for protecting circuitry are shown in Figure 26. A transient suppressor is placed in front of the regulator chip to protect it. Since the rise times in these transients are not like those in ESD pulses, lead inductance is less critical and conventional devices can be used. The regulator itself is pretty much of a necessity, since a load dump transient is simply not going to be removed

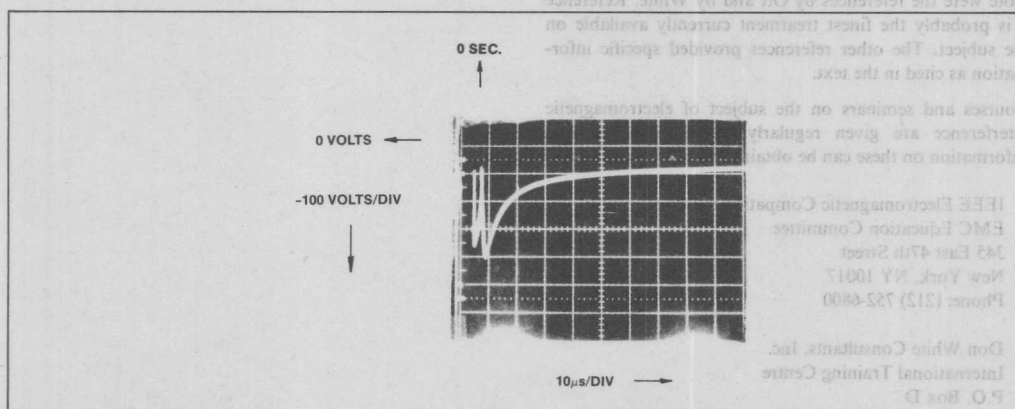


Figure 25. Transient Created by De-energizing an Air Conditioning Clutch Solenoid

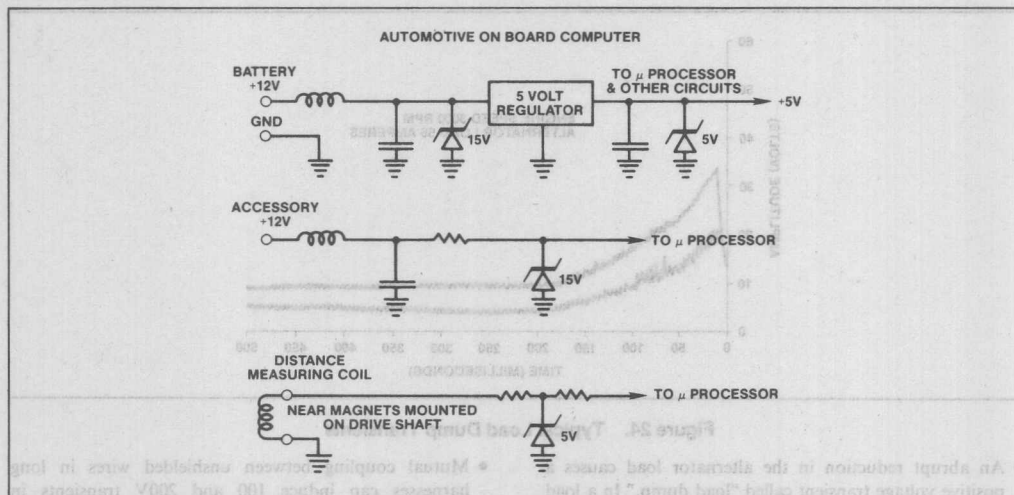


Figure 26. Use of Transient Suppressors in Automotive Applications

by any conventional LC or RC filter. Special I/O interfacing is also required, because of the need for high tolerance to voltage transients, input noise, input/output isolation, etc. In addition, switches that are being monitored or driven by these buffers are usually referenced to chassis ground instead of signal ground, and in a car there can be many volts difference between the two. I/O interfacing is discussed in reference 2.

### Parting Thoughts

The main sources of information for this Application Note were the references by Ott and by White. Reference 5 is probably the finest treatment currently available on the subject. The other references provided specific information as cited in the text.

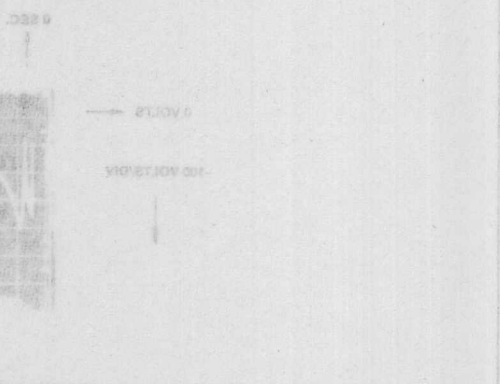
Courses and seminars on the subject of electromagnetic interference are given regularly throughout the year. Information on these can be obtained from:

IEEE Electromagnetic Compatibility Society  
EMC Education Committee  
345 East 47th Street  
New York, NY 10017  
Phone: (212) 752-6800

Don White Consultants, Inc.  
International Training Centre  
P.O. Box D  
Gainesville, VA 22065  
Phone: (703) 347-0030

The EMC Education committee has available a video tape: "Introduction to EMC — A Video Training Tape," by Henry Ott. Don White Consultants offers a series of training courses on many different aspects of electromagnetic compatibility. Most organizations that sponsor EMC courses also offer in-plant presentations.

When the ignition is turned off, as the field excitation decays, the line voltage can go to between -40 and -100V for 100 msec or more. Mischievous solenoid switching transients such as the one shown in Figure 25, can drive the line to +300 to 400V for several msec.



June 1983

for Microcontrollers  
Oscillators

Tom Williamson  
Microcontroller  
Technical Marketing

June 1983

**Oscillators  
for Microcontrollers**

**Tom Williamson**  
Microcontroller  
Technical Marketing

Order Number: 230659-001



# Oscillators for Microcontrollers

manipulate how oscillators work, how crystals and ceramic resonators work (and thus how to spec them), and what the on-chip amplifier looks like electronically, and what its operating characteristics are. A BASIC program is provided in Appendix II to assist the designer in determining the effects of changing individual parameters. Suggestions are provided for establishing a pre-production test program.

## FEEDBACK OSCILLATORS

### Loop Gain

Figure 1 shows an amplifier whose output line goes into some passive network. If the input signal to the amplifier is  $v_i$ , then the output signal from the amplifier is  $v_o = A v_i$ , and the output signal from the passive network is  $v_f = B v_o$ . Thus  $B A$  is the overall gain from terminal 1 to terminal 3.

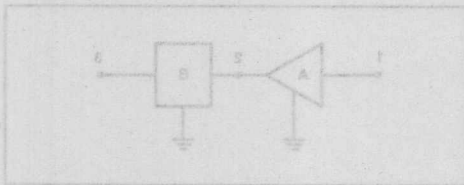


Figure 1 — Factors in Loop Gain

Now connect terminal 1 to terminal 3, so that the signal path forms a loop. 1 to 3, which is also 1. Now we have a feedback loop, and the gain factor  $B A$  is called the loop gain.

Gain factors are complex numbers. That means they have a magnitude and a phase angle, both of which vary with frequency. When writing a complex number, one must specify both magnitude, magnitude and angle. A number whose magnitude is 3, and whose angle is  $45^\circ$  is written in this way:  $3 \angle 45^\circ$ . The number 1 is in complex number notation,  $1 \angle 0^\circ$ , while  $-1$  is  $1 \angle 180^\circ$ .

By closing the feedback loop in Figure 0, we force the equality

$$v_i = B A v_i$$

This equation has two solutions:

$$1) v_i = 0$$

$$2) B A = 1$$

In a given circuit, either or both of the above solutions may be in effect. In the first solution the circuit is quiet (no output signal). If you're trying to make an

## CONTENTS

### INTRODUCTION ..... 22-25

### FEEDBACK OSCILLATORS

|   |       |
|---|-------|
| Loop Gain .....                         | 22-25 |
| How Feedback Oscillators Work .....     | 22-26 |
| The Positive Reactance Oscillator ..... | 22-26 |

### QUARTZ CRYSTALS

|  |       |
|--|-------|
| Crystal Parameters .....               | 22-27 |
| equivalent circuit .....               | 22-27 |
| load capacitance .....                 | 22-27 |
| "series" vs. "parallel" crystals ..... | 22-28 |
| equivalent series resistance .....     | 22-28 |
| frequency tolerance .....              | 22-28 |
| drive level .....                      | 22-29 |

### CERAMIC RESONATORS

|   |       |
|---|-------|
| Specifications for Ceramic Resonators ..... | 22-29 |
| Resonators .....                            | 22-30 |

### OSCILLATOR DESIGN CONSIDERATIONS

|   |       |
|---|-------|
| On-Chip Oscillators .....                   | 22-30 |
| crystal specifications .....                | 22-30 |
| oscillation frequency .....                 | 22-30 |
| selection of CX1 and CX2 .....              | 22-31 |
| placement of components .....               | 22-31 |
| clocking other chips .....                  | 22-31 |
| External Oscillators .....                  | 22-31 |
| gate oscillators vs. discrete devices ..... | 22-33 |
| fundamental vs. overtone operation .....    | 22-33 |
| "series" vs. "parallel" operation .....     | 22-33 |

### MORE ABOUT USING THE "ON-CHIP" OSCILLATORS

|   |       |
|---|-------|
| Oscillator Calculations .....             | 22-34 |
| Start-Up Characteristics .....            | 22-35 |
| Steady-State Characteristics .....        | 22-37 |
| Pin Capacitance .....                     | 22-38 |
| MCS*51 Oscillator .....                   | 22-39 |
| MCS*48 Oscillator .....                   | 22-39 |
| Pre-Production Tests .....                | 22-42 |
| troubleshooting oscillator problems ..... | 22-43 |

### APPENDIX I

|   |       |
|---|-------|
| Quartz and Ceramic Resonator Formulas ..... | 22-46 |
|---|-------|

### APPENDIX II

|                                   |       |
|-----------------------------------|-------|
| Oscillator Analysis Program ..... | 22-48 |
|-----------------------------------|-------|

## INTRODUCTION

Intel's microcontroller families (MCS®-48, MCS-51, and iACX-96) contain a circuit that is commonly referred to as the "on-chip oscillator". The on-chip circuitry is not itself an oscillator, of course, but an amplifier that is suitable for use as the amplifier part of a feedback oscillator. The data sheets and Microcontroller Handbook show how the on-chip amplifier and several off-chip components can be used to design a working oscillator. With proper selection of off-chip components, these oscillator circuits will perform better than almost any other type of clock oscillator, and by almost any criterion of excellence. The suggested circuits are simple, economical, stable, and reliable.

We offer assistance to our customers in selecting suitable off-chip components to work with the on-chip oscillator circuitry. It should be noted, however, that Intel cannot assume the responsibility of writing specifications for the off-chip components of the complete oscillator circuit, nor of guaranteeing the performance of the finished design in production, anymore than a transistor manufacturer, whose data sheets show a number of suggested amplifier circuits, can assume responsibility for the operation, in production, of any of them.

We are often asked why we don't publish a list of required crystal or ceramic resonator specifications, and recommend values for the other off-chip components. This has been done in the past, but sometimes with consequences that were not intended.

Suppose we suggest a maximum crystal resistance of 30 ohms for some given frequency. Then your crystal supplier tells you the 30-ohm crystals are going to cost twice as much as 50-ohm crystals. Fearing that Intel will not "guarantee operation" with 50-ohm crystals, you order the expensive ones. In fact, Intel guarantees only what is embodied within an Intel product. Besides, there is no reason why 50-ohm crystals couldn't be used, if the other off-chip components are suitably adjusted.

Should we recommend values for the other off-chip components? Should we do it for 50-ohm crystals or 30-ohm crystals? With respect to what should we optimize their selection? Should we minimize start-up time or maximize frequency stability? In many applications, neither start-up time nor frequency stability are particularly critical, and our "recommendations" are only restricting your system to unnecessary tolerances. It all depends on the application.

Although we will neither "specify" nor "recommend" specific off-chip components, we do offer assistance in these tasks. Intel applications engineers are available to provide whatever technical assistance may be needed or desired by our customers in designing with Intel products.

This Application Note is intended to provide such assistance

in the design of oscillator circuits for microcontroller systems. Its purpose is to describe in a practical manner how oscillators work, how crystals and ceramic resonators work (and thus how to spec them), and what the on-chip amplifier looks like electronically and what its operating characteristics are. A BASIC program is provided in Appendix II to assist the designer in determining the effects of changing individual parameters. Suggestions are provided for establishing a pre-production test program.

## FEEDBACK OSCILLATORS

### Loop Gain

Figure 1 shows an amplifier whose output line goes into some passive network. If the input signal to the amplifier is  $v_1$ , then the output signal from the amplifier is  $v_2 = Av_1$ , and the output signal from the passive network is  $v_3 = \beta v_2 = \beta Av_1$ . Thus  $\beta A$  is the overall gain from terminal 1 to terminal 3.

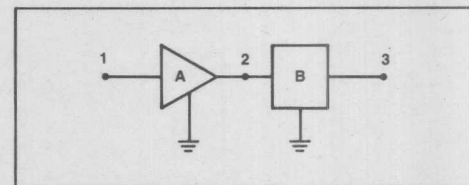


Figure 1 — Factors in Loop Gain

Now connect terminal 1 to terminal 3, so that the signal path forms a loop: 1 to 2 to 3, which is also 1. Now we have a feedback loop, and the gain factor  $\beta A$  is called the *loop gain*.

Gain factors are complex numbers. That means they have a magnitude and a phase angle, both of which vary with frequency. When writing a complex number, one must specify both quantities, magnitude and angle. A number whose magnitude is 3, and whose angle is 45 degrees is commonly written this way:  $3/45^\circ$ . The number 1 is, in complex number notation,  $1/0^\circ$ , while  $-1$  is  $1/180^\circ$ .

By closing the feedback loop in Figure 0, we force the equality

$$v_1 = \beta Av_1$$

This equation has two solutions:

- 1)  $v_1 = 0$ ;
- 2)  $\beta A = 1/0^\circ$ .

In a given circuit, either or both of the above solutions may be in effect. In the first solution the circuit is quiescent (no output signal). If you're trying to make an

oscillator, a no-signal condition is unacceptable. There are ways to guarantee that the second solution is the one that will be in effect, and that the quiescent condition will be excluded.

### How Feedback Oscillators Work

A feedback oscillator amplifies its own noise and feeds it back to itself in exactly the right phase, at the oscillation frequency, to build up and reinforce the desired oscillations. Its ability to do that depends on its loop gain. First, oscillations can occur only at the frequency for which the loop gain has a phase angle of 0 degrees. Second, build-up of oscillations will occur only if the loop gain exceeds 1 at that frequency. Build-up continues until nonlinearities in the circuit reduce the average value of the loop gain to exactly 1.

Start-up characteristics depend on the small-signal properties of the circuit, specifically, the small-signal loop gain. Steady-state characteristics of the oscillator depend on the large-signal properties of the circuit, such as the transfer curve (output voltage vs. input voltage) of the amplifier, and the clamping effect of the input protection devices. These things will be discussed more fully further on. First we will look at the basic operation of a particular oscillator circuit, called the "positive reactance" oscillator.

### The Positive Reactance Oscillator

Figure 2 shows the configuration of the positive reactance oscillator. The inverting amplifier, working into the impedance of the feedback network, produces an output signal that is nominally 180 degrees out of phase with its input. The feedback network must provide an additional 180 degrees phase shift, such that the overall loop gain has zero (or 360) degrees phase shift at the oscillation frequency.

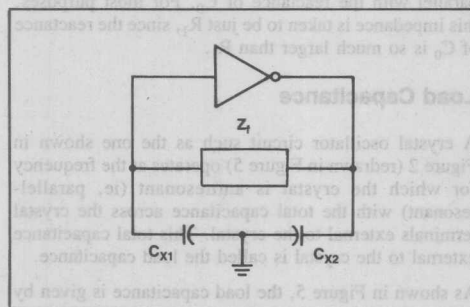


Figure 2 — Positive Reactance Oscillator

In order for the loop gain to have zero phase angle it is necessary that the feedback element  $Z_f$  have a positive

reactance. That is, it must be inductive. Then, the frequency at which the phase angle is zero is approximately the frequency at which

$$X_f = \frac{+1}{\omega C}$$

where  $X_f$  is the reactance of  $Z_f$  (the total  $Z_f$  being  $R_f + jX_f$ , and  $C$  is the series combination of  $C_{x1}$  and  $C_{x2}$ ).

$$C = \frac{C_{x1}C_{x2}}{C_{x1} + C_{x2}}$$

In other words,  $Z_f$  and  $C$  form a parallel resonant circuit.

If  $Z_f$  is an inductor, then  $X_f = \omega L$ , and the frequency at which the loop gain has zero phase is the frequency at which

$$\omega L = \frac{1}{\omega C}$$

or

$$\omega = \frac{1}{\sqrt{LC}}$$

Normally,  $Z_f$  is not an inductor, but it must still have a positive reactance in order for the circuit to oscillate. There are some piezoelectric devices on the market that show a positive reactance, and provide a more stable oscillation frequency than an inductor will. Quartz crystals can be used where the oscillation frequency is critical, and lower cost ceramic resonators can be used where the frequency is less critical.

When the feedback element is a piezoelectric device, this circuit configuration is called a Pierce oscillator. The advantage of piezoelectric resonators lies in their property of providing a wide range of positive reactance values over a very narrow range of frequencies. The reactance will equal  $1/\omega C$  at some frequency within this range, so the oscillation frequency will be within the same range. Typically, the width of this range is only .3% of the nominal frequency of a quartz crystal, and about 3% of the nominal frequency of a ceramic resonator. With relatively little design effort, frequency accuracies of .03% or better can be obtained with quartz crystals, and .3% or better with ceramic resonators.

### QUARTZ CRYSTALS

The crystal resonator is a thin slice of quartz sandwiched between two electrodes. Electrically, the device looks pretty much like a 5 or 6 pF capacitor, except that over certain ranges of frequencies the crystal has a positive (i.e., inductive) reactance.

The ranges of positive reactance originate in the piezoelectric property of quartz: Squeezing the crystal gener-

ates an internal E-field. The effect is reversible: Applying an E-field causes a mechanical deflection. Applying an AC E-field causes the crystal to vibrate. At certain vibrational frequencies there is a mechanical resonance. As the E-field frequency approaches a frequency of mechanical resonance, the measured reactance of the crystal becomes positive, as shown in Figure 3.

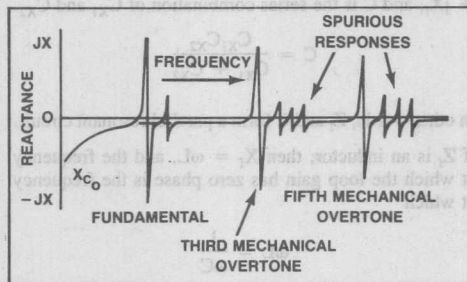


Figure 3 — Crystal Reactance vs. Frequency

Typically there are several ranges of frequencies wherein the reactance of the crystal is positive. Each range corresponds to a different mode of vibration in the crystal. The main resonances are the so-called fundamental response and the third and fifth overtone responses.

The overtone responses shouldn't be confused with the harmonics of the fundamental. They're not harmonics, but different vibrational modes. They're not in general at exact integer multiples of the fundamental frequency. There will also be "spurious" responses, occurring typically a few hundred KHz above each main response.

To assure that an oscillator starts in the desired mode on power-up, something must be done to suppress the loop gain in the undesired frequency ranges. The crystal itself provides some protection against unwanted modes of oscillation; too much resistance in that mode, for example. Additionally, junction capacitances in the amplifying devices tend to reduce the gain at higher frequencies, and thus may discriminate against unwanted modes. In some cases a circuit fix is necessary, such as inserting a trap, a phase shifter, or ferrite beads to kill oscillations in unwanted modes.

## Crystal Parameters

### Equivalent Circuit

Figure 4 shows an equivalent circuit that is used to represent the crystal for circuit analysis.

The  $R_1$ - $L_1$ - $C_1$  branch is called the motional arm of the crystal. The values of these parameters derive from the mechanical properties of the crystal and are constant for a given mode of vibration. Typical values for various nominal frequencies are shown in Table 1.

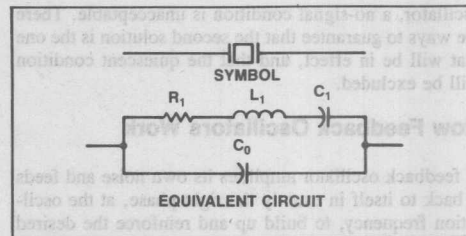


Figure 4 — Quartz Crystal:  
Symbol and Equivalent Circuit

$C_0$  is called the shunt capacitance of the crystal. This is the capacitance of the crystal's electrodes and the mechanical holder. If one were to measure the reactance of the crystal at a frequency far removed from a resonance frequency, it is the reactance of this capacitance that would be measured. It's normally 3 to 7 pF.

Table 1 — Typical Crystal Parameters

| frequency<br>MHz | $R_1$<br>ohms | $L_1$<br>mH | $C_1$<br>pF | $C_0$<br>pF |
|------------------|---------------|-------------|-------------|-------------|
| 2                | 100           | 520         | .012        | 4           |
| 4.608            | 36            | 117         | .010        | 2.9         |
| 11.25            | 19            | 8.38        | .024        | 5.4         |

The series resonant frequency of the crystal is the frequency at which  $L_1$  and  $C_1$  are in resonance. This frequency is given by

$$f_s = \frac{1}{2\pi \sqrt{L_1 C_1}}$$

At this frequency the impedance of the crystal is  $R_1$  in parallel with the reactance of  $C_0$ . For most purposes, this impedance is taken to be just  $R_1$ , since the reactance of  $C_0$  is so much larger than  $R_1$ .

### Load Capacitance

A crystal oscillator circuit such as the one shown in Figure 2 (redrawn in Figure 5) operates at the frequency for which the crystal is antiresonant (ie, parallel-resonant) with the total capacitance across the crystal terminals external to the crystal. This total capacitance external to the crystal is called the load capacitance.

As shown in Figure 5, the load capacitance is given by

$$C_L = \frac{C_{X1} C_{X2}}{C_{X1} + C_{X2}} + C_{\text{stray}}$$

The crystal manufacturer needs to know the value of  $C_L$  in order to adjust the crystal to the specified frequency.



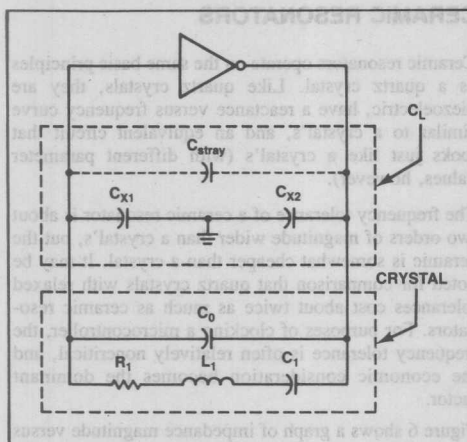


Figure 5 — Load Capacitance

The adjustment involves putting the crystal in series with the specified  $C_L$ , and then "trimming" the crystal to obtain resonance of the series combination of the crystal and  $C_L$  at the specified frequency. Because of the high  $Q$  of the crystal, the resonant frequency of the series combination of the crystal and  $C_L$  is the same as the antiresonant frequency of the parallel combination of the crystal and  $C_L$ . This frequency is given by

$$f_a = \frac{1}{2\pi\sqrt{L_1 C_1 (C_L + C_0) / (C_1 + C_L + C_0)}}$$

These frequency formulas are derived (in Appendix I) from the equivalent circuit of the crystal, using the assumptions that the  $Q$  of the crystal is extremely high, and that the circuit external to the crystal has no effect on the frequency other than to provide the load capacitance  $C_L$ . The latter assumption is not precisely true, but it is close enough for present purposes.

### "Series" vs. "Parallel" Crystals

There is no such thing as a "series cut" crystal as opposed to a "parallel cut" crystal. There are different cuts of crystal, having to do with the parameters of its motional arm in various frequency ranges, but there is no special cut for series or parallel operation.

An oscillator is series resonant if the oscillation frequency is  $f_s$  of the crystal. To operate the crystal at  $f_s$ , the amplifier has to be noninverting. When buying a crystal for such an oscillator, one does not specify a load capacitance. Rather, one specifies the loading condition as "series."

If a "series" crystal is put into an oscillator that has an inverting amplifier, it will oscillate in parallel resonance with the load capacitance presented to the crystal by the

oscillator circuit, at a frequency slightly above  $f_s$ . In fact, at approximately

$$f_a = f_s \left( 1 + \frac{C_1}{2(C_L + C_0)} \right)$$

This frequency would typically be about 0.02% above  $f_s$ .

### Equivalent Series Resistance

The "series resistance" often listed on quartz crystal data sheets is the real part of the crystal impedance at the crystal's calibration frequency. This will be  $R_1$  if the calibration frequency is the series resonant frequency of the crystal. If the crystal is calibrated for parallel resonance with a load capacitance  $C_L$ , the equivalent series resistance will be

$$ESR = R_1 \left( 1 + \frac{C_0}{C_L} \right)^2$$

The crystal manufacturer measures this resistance at the calibration frequency during the same operation in which the crystal is adjusted to the calibration frequency.

### Frequency Tolerance

Frequency tolerance as discussed here is not a requirement on the crystal, but on the complete oscillator. There are two types of frequency tolerances on oscillators: frequency accuracy and frequency stability. Frequency accuracy refers to the oscillator's ability to run at an exact specified frequency. Frequency stability refers to the constancy of the oscillation frequency.

Frequency accuracy requires mainly that the oscillator circuit present to the crystal the same load capacitance that it was adjusted for. Frequency stability requires mainly that the load capacitance be constant.

In most digital applications the accuracy and stability requirements on the oscillator are so wide that it makes very little difference what load capacitance the crystal was adjusted to, or what load capacitance the circuit actually presents to the crystal. For example, if a crystal was calibrated to a load capacitance of 25 pF, and is used in a circuit whose actual load capacitance is 50 pF, the frequency error on that account would be less than 0.01%.

In a positive reactance oscillator, the crystal only needs to be in the intended response mode for the oscillator to satisfy a 0.5% or better frequency tolerance. That's because for any load capacitance the oscillation frequency is certain to be between the crystal's resonant and antiresonant frequencies.

Phase shifts that take place within the amplifier part of the oscillator will also affect frequency accuracy and



stability. These phase shifts can normally be modeled as an "output capacitance" that, in the positive reactance oscillator, parallels  $C_{X2}$ . The predictability and constancy of this output capacitance over temperature and device sample will be the limiting factor in determining the tolerances that the circuit is capable of holding.

## Drive Level

Drive level refers to the power dissipation in the crystal. There are two reasons for specifying it. One is that the parameters in the equivalent circuit are somewhat dependent on the drive level at which the crystal is calibrated. The other is that if the application circuit exceeds the test drive level by too much, the crystal may be damaged. Note that the terms "test drive level" and "rated drive level" both refer to the drive level at which the crystal is calibrated. Normally, in a microcontroller system, neither the frequency tolerances nor the power levels justify much concern for this specification. Some crystal manufacturers don't even require it for microprocessor crystals.

In a positive reactance oscillator, if one assumes the peak voltage across the crystal to be something in the neighborhood of  $V_{CC}$ , the power dissipation can be approximated as

$$P = 2R_1[\pi f (C_L + C_0) V_{CC}]^2$$

This formula is derived in Appendix I. In a 5V system, P rarely evaluates to more than a milliwatt. Crystals with a standard 1 or 2 mW drive level rating can be used in most digital systems.

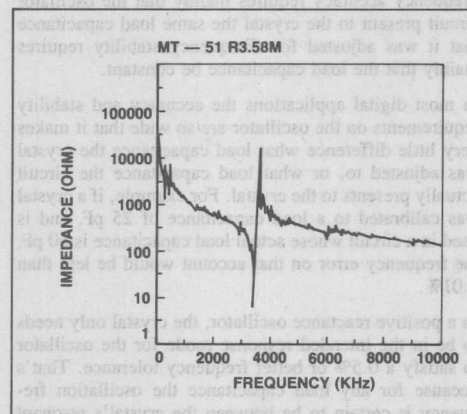


Figure 6 — Ceramic Resonator Impedance vs. Frequency (Test Data Supplied by NTK Technical Ceramics)

## CERAMIC RESONATORS

Ceramic resonators operate on the same basic principles as a quartz crystal. Like quartz crystals, they are piezoelectric, have a reactance versus frequency curve similar to a crystal's, and an equivalent circuit that looks just like a crystal's (with different parameter values, however).

The frequency tolerance of a ceramic resonator is about two orders of magnitude wider than a crystal's, but the ceramic is somewhat cheaper than a crystal. It may be noted for comparison that quartz crystals with relaxed tolerances cost about twice as much as ceramic resonators. For purposes of clocking a microcontroller, the frequency tolerance is often relatively noncritical, and the economic consideration becomes the dominant factor.

Figure 6 shows a graph of impedance magnitude versus frequency for a 3.58MHz ceramic resonator. (Note that Figure 6 is a graph of  $|Z|$  versus frequency, whereas Figure 3 is a graph of  $X_r$  versus frequency.) A number of spurious responses are apparent in Figure 6. The manufacturers state that spurious responses are more prevalent in the lower frequency resonators (kHz range) than in the higher frequency units (MHz range). For our purposes only the MHz range ceramics need to be considered.

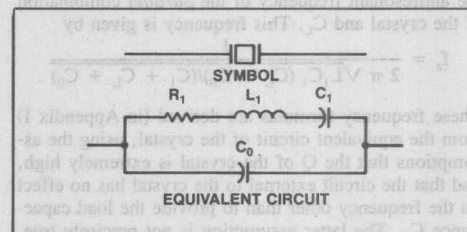


Figure 7 — Ceramic Resonator: Symbol and Equivalent Circuit

Figure 7 shows the symbol and equivalent circuit for the ceramic resonator, both of which are the same as for the crystal. The parameters have different values, however, as listed in Table 2.

Table 2 — Typical Ceramic Parameters

| frequency<br>MHz | $R_1$<br>ohms | $L_1$<br>mH | $C_1$<br>pF | $C_0$<br>pF |
|------------------|---------------|-------------|-------------|-------------|
| 3.58             | 7             | .113        | 19.6        | 140         |
| 6.0              | 8             | .094        | 8.3         | 60          |
| 8.0              | 7             | .092        | 4.6         | 40          |
| 11.0             | 10            | .057        | 3.9         | 30          |

Note that the motional arm of the ceramic resonator tends to have less resistance than the quartz crystal and also a vastly reduced  $L_1/C_1$  ratio. This results in the motional arm having a  $Q$  (given by  $(1/R_1) \sqrt{L_1/C_1}$ ) that is typically two orders of magnitude lower than that of a quartz crystal. The lower  $Q$  makes for a faster start-up of the oscillator and for a less closely controlled frequency (meaning that circuitry external to the resonator will have more influence on the frequency than with a quartz crystal).

Another major difference is that the shunt capacitance of the ceramic resonator is an order of magnitude higher than  $C_0$  of the quartz crystal and more dependent on the frequency of the resonator.

The implications of these differences are not all obvious, but some will be indicated in the section on Oscillator Calculations.

### Specifications for Ceramic Resonators

Ceramic resonators are easier to specify than quartz crystals. All the vendor wants to know is the desired frequency and the chip you want it to work with. They'll supply the resonators, a circuit diagram showing the positions and values of other external components that may be required and a guarantee that the circuit will work properly at the specified frequency.

## OSCILLATOR DESIGN CONSIDERATIONS

Designers of microcontroller systems have a number of options to choose from for clocking the system. The main decision is whether to use the "on-chip" oscillator or an external oscillator. If the choice is to use the on-chip oscillator, what kinds of external components are needed to make it operate as advertised? If the choice is to use an external oscillator, what type of oscillator should it be?

The decisions have to be based on both economic and technical requirements. In this section we'll discuss some of the factors that should be considered.

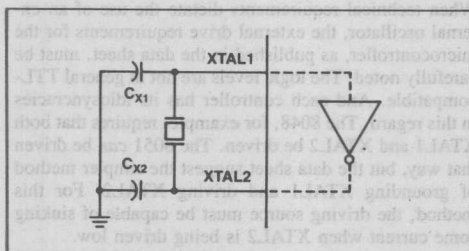


Figure 8 — Using the "On-Chip" Oscillator

## On-Chip Oscillators

In most cases, the on-chip amplifier with the appropriate external components provides the most economical solution to the clocking problem. Exceptions may arise in severe environments when frequency tolerances are tighter than about 0.01%.

The external components that need to be added are a positive reactance (normally a crystal or ceramic resonator) and the two capacitors  $C_{X1}$  and  $C_{X2}$ , as shown in Figure 8.

## Crystal Specifications

Specifications for an appropriate crystal are not very critical, unless the frequency is. Any fundamental-mode crystal of medium or better quality can be used.

We are often asked what maximum crystal resistance should be specified. The best answer to this question is the lower the better, but use what's available. The crystal resistance will have some effect on start-up time and steady-state amplitude, but not so much that it can't be compensated for by appropriate selection of the capacitances  $C_{X1}$  and  $C_{X2}$ .

Similar questions are asked about specifications of load capacitance and shunt capacitance. The best advice we can give is to understand what these parameters mean and how they affect the operation of the circuit (that being the purpose of this Application Note), and then decide for yourself if such specifications are meaningful in your application or not. Normally, they're not, unless your frequency tolerances are tighter than about 0.1%.

Part of the problem is that crystal manufacturers are accustomed to talking "ppm" tolerances with radio engineers and simply won't take your order until you've filled out their list of specifications. It will help if you define your actual frequency tolerance requirements, both for yourself and to the crystal manufacturer. Don't pay for 0.003% crystals if your actual frequency tolerance is 1%.

## Oscillation Frequency

The oscillation frequency is determined 99.5% by the crystal and up to about 0.5% by the circuit external to the crystal. The on-chip amplifier has little effect on the frequency, which is as it should be, since the amplifier parameters are temperature and process dependent.

The influence of the on-chip amplifier on the frequency is by means of its input and output (pin-to-ground) capacitances, which parallel  $C_{X1}$  and  $C_{X2}$ , and the XTAL1-to-XTAL2 (pin-to-pin) capacitance, which parallels the crystal. The input and pin-to-pin capacitances are about 7 pF each. Internal phase deviations from the nominal 180° can be modeled as an output capacitance of 25 to 30 pF. These deviations from the ideal have less effect

in the positive reactance oscillator (with the inverting amplifier) than in a comparable series resonant oscillator (with the noninverting amplifier) for two reasons: first, the effect of the output capacitance is lessened, if not swamped, by the off-chip capacitor; secondly, the positive reactance oscillator is less sensitive, frequency-wise, to such phase errors.

### Selection of $C_{X1}$ and $C_{X2}$

Optimal values for the capacitors  $C_{X1}$  and  $C_{X2}$  depend on whether a quartz crystal or ceramic resonator is being used, and also on application-specific requirements on start-up time and frequency tolerance.

Start-up time is sometimes more critical in microcontroller systems than frequency stability, because of various reset and initialization requirements.

Less commonly, accuracy of the oscillator frequency is also critical, for example, when the oscillator is being used as a time base. As a general rule, fast start-up and stable frequency tend to pull the oscillator design in opposite directions.

Considerations of both start-up time and frequency stability over temperature suggest that  $C_{X1}$  and  $C_{X2}$  should be about equal and at least 20 pF. (But they don't have to be either.) Increasing the value of these capacitances above some 40 or 50 pF improves frequency stability. It also tends to increase the start-up time. There is a maximum value (several hundred pF, depending on the value of  $R_1$  of the quartz or ceramic resonator) above which the oscillator won't start up at all.

If the on-chip amplifier is a simple inverter, such as in the 8051, the user can select values for  $C_{X1}$  and  $C_{X2}$  between some 20 and 100 pF, depending on whether start-up time or frequency stability is the more critical parameter in a specific application. If the on-chip amplifier is a Schmitt Trigger, such as in the 8048, smaller values of  $C_{X1}$  must be used (5 to 30 pF), in order to prevent the oscillator from running in a relaxation mode.

Later sections in this Application Note will discuss the effects of varying  $C_{X1}$  and  $C_{X2}$  (as well as other parameters), and will have more to say on their selection.

### Placement of Components

Noise glitches arriving at the XTAL1 or XTAL2 pins at the wrong time can cause a miscount in the internal clock-generating circuitry. These kinds of glitches can be produced through capacitive coupling between the oscillator components and PCB traces carrying digital signals with fast rise and fall times. For this reason, the oscillator components should be mounted close to the chip and have short, direct traces to the XTAL1, XTAL2, and VSS pins.

### Clocking Other Chips

There are times when it would be desirable to use the on-chip oscillator to clock other chips in the system.

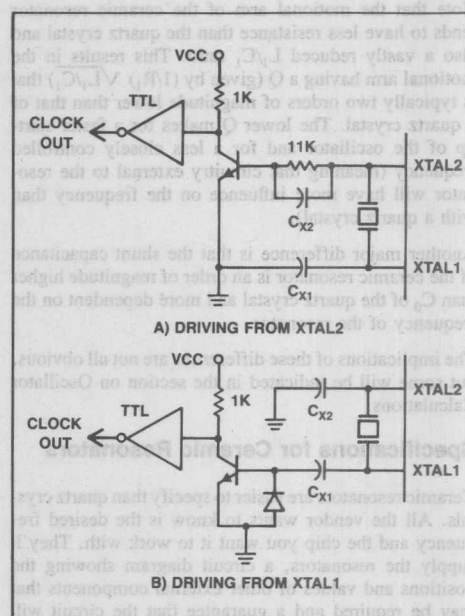


Figure 9 — Using the On-Chip Oscillator to Drive Other Chips

This can be done if an appropriate buffer is used. A TTL buffer puts too much load on the on-chip amplifier for reliable start-up. A CMOS buffer (such as the 74HC04) can be used, if it's fast enough and if its  $V_{IH}$  and  $V_{IL}$  specs are compatible with the available signal amplitudes. Circuits such as shown in Figure 9 might also be considered for these types of applications.

Clock-related signals are available at the TO pin in the MCS-48 products, at ALE in the MCS-48 and MCS-51 lines, and the iACX-96 controllers provide a CLKOUT signal.

### External Oscillators

When technical requirements dictate the use of an external oscillator, the external drive requirements for the microcontroller, as published in the data sheet, must be carefully noted. The logic levels are not in general TTL-compatible. And each controller has its idiosyncracies in this regard. The 8048, for example, requires that both XTAL1 and XTAL2 be driven. The 8051 can be driven that way, but the data sheet suggest the simpler method of grounding XTAL1 and driving XTAL2. For this method, the driving source must be capable of sinking some current when XTAL2 is being driven low.

For the external oscillator itself, there are basically two choices: ready-made and home-grown.

**TTL Crystal Clock Oscillator**

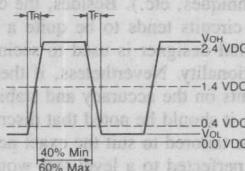
The HS-100, HS-200, & HS-500 all-metal package series of oscillators are TTL compatible & fit a DIP layout. Standard electrical specifications are shown below. Variations are available for special applications.

**Frequency Range:** HS-100 — 3.5 MHz to 30 MHz  
 HS-200 — 225 KHz to 3.5 MHz  
 HS-500 — 25 MHz to 60 MHz

**Frequency Tolerance:**  $\pm 0.01\%$  Overall 0-70° C

**Hermetically Sealed Package**

Mass spectrometer leak rate max.  
 $1 \times 10^{-9}$  atmos. cc/sec. of helium

**OUTPUT WAVE FORM**

|  | INPUT  |                                |                                |                                |
|--|--|--------------------------------|--------------------------------|--------------------------------|
|  | HS-100   |                                | HS-200                         | HS-500                         |
|  | 3.5 MHz - 20 MHz                               | 20+ MHz - 30 MHz               | 225 KHz - 4.0 MHz              | 25 MHz - 60 MHz                |
| Supply Voltage (V <sub>CC</sub> )                      | 5V $\pm$ 10%                                   | 5V $\pm$ 10%                   | 5V $\pm$ 10%                   | 5V $\pm$ 5%                    |
| Supply Current (I <sub>CC</sub> ) max.                 | 30mA   | 40mA                           | 85mA                           | 50mA                           |
|  | OUTPUT   |                                |                                |                                |
|  | HS-100   |                                | HS-200                         | HS-500                         |
|  | 3.5 MHz - 20 MHz                               | 20+ MHz - 30 MHz               | 225 KHz - 4.0 MHz              | 25 MHz - 60 MHz                |
| V <sub>OH</sub> (Logic "1")                            | +2.4V min. <sup>1</sup>                        | +2.7V min. <sup>2</sup>        | +2.4V min. <sup>3</sup>        | +2.7V min. <sup>2</sup>        |
| V <sub>OL</sub> (Logic "0")                            | +0.4V max. <sup>3</sup>                        | +0.5V max. <sup>4</sup>        | +0.4V max. <sup>3</sup>        | +0.5V max. <sup>4</sup>        |
| Symmetry   | 60/40% <sup>5</sup>                            | 60/40% <sup>5</sup>            | 55/45% <sup>5</sup>            | 60/40% <sup>5</sup>            |
| T <sub>R</sub> , T <sub>F</sub> (Rise & Fall Time)     | < 10ns <sup>6</sup>                            | < 5ns <sup>6</sup>             | < 15ns <sup>6</sup>            | < 5ns <sup>6</sup>             |
| Output Short   |  |                                |                                |                                |
| Circuit Current  | 18mA min.                                      | 40mA min.                      | 18mA min.                      | 40mA min.                      |
| Output Load  | 1 to 10 TTL Loads <sup>7</sup>                 | 1 to 10 TTL Loads <sup>8</sup> | 1 to 10 TTL Loads <sup>7</sup> | 1 to 10 TTL Loads <sup>8</sup> |
| <b>CONDITIONS</b>                                      |  |                                |                                |                                |
| <sup>1</sup> I <sub>Q</sub> source = -400 $\mu$ A max. | <sup>4</sup> I <sub>Q</sub> sink = 20.0mA max. | <sup>7</sup> 1.6mA per load    |                                |                                |
| <sup>2</sup> I <sub>Q</sub> source = -1.0mA max.       | <sup>5</sup> V <sub>O</sub> = 1.4V             | <sup>8</sup> 2.0mA per load    |                                |                                |
| <sup>3</sup> I <sub>Q</sub> sink = 16.0mA max.         | <sup>6</sup> (0.4V to 2.4V)                    |                                |                                |                                |

**Figure 10 — Pre-packaged Oscillator Data\***

Prepackaged oscillators are available from most crystal manufacturers, and have the advantage that the system designer can treat the oscillator as a black box whose performance is guaranteed by people who carry many years of experience in designing and building oscillators. Figure 10 shows a typical data sheet for some prepackaged oscillators. Oscillators are also available with complementary outputs.

If the oscillator is to drive the microcontroller directly, one will want to make a careful comparison between the external drive requirements in the microcontroller data sheet and the oscillator's output logic levels and test conditions.

If oscillator stability is less critical than cost, the user

may prefer to go with an in-house design. Not without some precautions, however.

It's easy to design oscillators that work. Almost all of them do work, even if the designer isn't too clear on why. The key point here is that *almost* all of them work. The problems begin when the system goes into production, and marginal units commence malfunctioning in the field. Most digital designers, after all, are not very adept at designing oscillators for production.

Oscillator design is somewhat of a black art, with the quality of the finished product being very dependent on the designer's experience and intuition. For that reason the most important consideration in any design is to have an adequate preproduction test program. Preproduction tests are discussed later in this Application Note. Here we will discuss some of the design options and take a look at some commonly used configurations.

\*Reprinted with the permission of © Midland-Ross Corporation 1982



## Gate Oscillators versus Discrete Devices

Digital systems designers are understandably reluctant to get involved with discrete devices and their peculiarities (biasing techniques, etc.). Besides, the component count for these circuits tends to be quite a bit higher than what a digital designer is used to seeing for that amount of functionality. Nevertheless, if there are unusual requirements on the accuracy and stability of the clock frequency, it should be noted that discrete device oscillators can be tailored to suit the exact needs of the application and perfected to a level that would be difficult for a gate oscillator to approach.

In most cases, when an external oscillator is needed, the designer tends to rely on some form of a gate oscillator. A TTL inverter with a resistor connecting the output to the input makes a suitable inverting amplifier. The resistor holds the inverter in the transition region between logical high and low, so that at least for start-up purposes the inverter is a linear amplifier.

The feedback resistance has to be quite low, however, since it must conduct current sourced by the input pin without allowing the DC input voltage to get too far above the DC output voltage. For biasing purposes, the feedback resistance should not exceed a few k-ohms.

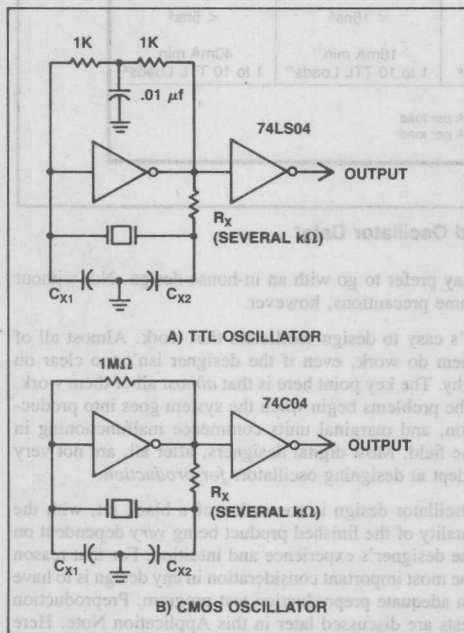


Figure 11 — Commonly Used Gate Oscillators

But shunting the crystal with such a low resistance does not encourage start-up.

Consequently, the configuration in Figure 11A might be suggested. By breaking  $R_f$  into two parts and AC-grounding the midpoint, one achieves the DC feedback required to hold the inverter in its active region, but without the negative signal feedback that is in effect telling the circuit *not* to oscillate. However, this biasing scheme will increase the start-up time, and relaxation-type oscillations are also possible.

A CMOS inverter, such as the 74HC04, might work better in this application, since a larger  $R_f$  can be used to hold the inverter in its linear region.

Logic gates tend to have a fairly low output resistance, which destabilizes the oscillator. For that reason a resistor  $R_x$  is often added to the feedback network, as shown in Figure 11 A and B. At higher frequencies a 20 or 30 pF capacitor is sometimes used in the  $R_x$  position, to compensate for some of the internal propagation delay.

Reference 1 contains an excellent discussion of gate oscillators, and a number of design examples.

## Fundamental versus Overtone Operation

It's easier to design an oscillator circuit to operate in the resonator's fundamental response mode than to design one for overtone operation. A quartz crystal whose fundamental response mode covers the desired frequency can be obtained up to some 30 MHz. For frequencies above that, the crystal might be used in an overtone mode.

Several problems arise in the design of an overtone oscillator. One is to stop the circuit from oscillating in the fundamental mode, which is what it would really rather do, for a number of reasons, involving both the amplifying device and the crystal. An additional problem with overtone operation is an increased tendency to spurious oscillations. That is because the  $R_1$  of various spurious modes is likely to be about the same as  $R_1$  of the intended overtone response. It may be necessary, as suggested in reference 1, to specify a "spurious-to-main-response" resistance ratio to avoid the possibility of trouble.

Overtone oscillators are not to be taken lightly. One would be well advised to consult with an engineer who is knowledgeable in the subject during the design phase of such a circuit.

## Series versus Parallel Operation

Series resonant oscillators use noninverting amplifiers. To make a noninverting amplifier out of logic gates requires that two inverters be used, as shown in Figure 12.

This type of circuit tends to be inaccurate and unstable



in frequency over variations in temperature and VCC. It has a tendency to oscillate at overtones, and to oscillate through  $C_0$  of the crystal or some stray capacitance rather than as controlled by the mechanical resonance of the crystal.

The demon in series resonant oscillators is the phase shift in the amplifier. The series resonant oscillator wants more than just a "noninverting" amplifier — it wants a zero phase-shift amplifier. Multistage noninverting amplifiers tend to have a considerably lagging phase shift, such that the crystal reactance must be capacitive in order to bring the total phase shift around the feedback loop back up to 0. In this mode, a "12 MHz" crystal may be running at 8 or 9 MHz. One can put a capacitor in series with the crystal to relieve the crystal of having to produce all of the required phase shift, and bring the oscillation frequency closer to  $f_s$ . However, to further complicate the situation, the amplifier's phase shift is strongly dependent on frequency, temperature, VCC, and device sample.

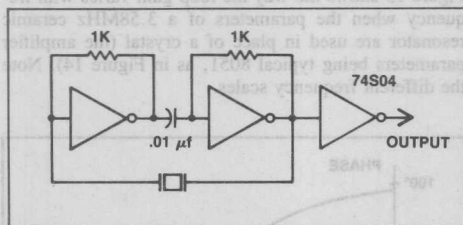


Figure 12 — "Series Resonant"  
Gate Oscillator

Positive reactance oscillators ("parallel resonant") use inverting amplifiers. A single logic inverter can be used for the amplifier, as in Figure 11. The amplifier's phase shift is less critical, compared to the series resonant circuit, and since only one inverter is involved there's less phase error anyway. The oscillation frequency is effectively bounded by the resonant and antiresonant frequencies of the crystal itself. In addition, the feedback network includes capacitors that parallel the input and output terminals of the amplifier, thus reducing the effect of unpredictable capacitances at these points.

### MORE ABOUT USING THE "ON-CHIP" OSCILLATORS

In this section we will describe the on-chip inverters on selected microcontrollers in some detail, and discuss criteria for selecting components to work with them. Future data sheets will supplement this discussion with updates and information pertinent to the use of each chip's oscillator circuitry.

### Oscillator Calculations

Oscillator design, though aided by theory, is still largely an empirical exercise. The circuit is inherently nonlinear, and the normal analysis parameters vary with instantaneous voltage. In addition, when dealing with the on-chip circuitry, we have FETs being used as resistors, resistors being used as interconnects, distributed delays, input protection devices, parasitic junctions, and processing variations.

Consequently, oscillator calculations are never very precise. They can be useful, however, if they will at least indicate the effects of variations in the circuit parameters on start-up time, oscillation frequency, and steady-state amplitude. Start-up time, for example, can be taken as an indication of start-up reliability. If preproduction tests indicate a possible start-up problem, a relatively inexperienced designer can at least be made aware of what

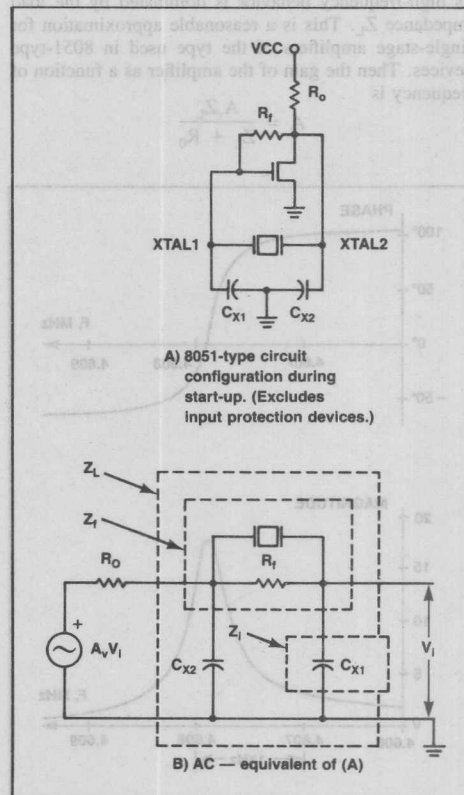


Figure 13 — Oscillator Circuit Model  
Used in Start-Up Calculations

parameter may be causing the marginality, and what direction to go in to fix it.

The analysis used here is mathematically straightforward but algebraically intractable. That means it's relatively easy to understand and program into a computer, but it will not yield a neat formula that gives, say, steady-state amplitude as a function of this or that list of parameters. A listing of a BASIC program that implements the analysis will be found in Appendix II.

When the circuit is first powered up, and before the oscillations have commenced (and if the oscillations fail to commence), the oscillator can be treated as a small signal linear amplifier with feedback. In that case, standard small-signal analysis techniques can be used to determine start-up characteristics. The circuit model used in this analysis is shown in Figure 13.

The circuit approximates that there are no high-frequency effects within the amplifier itself, such that its high-frequency behavior is dominated by the load impedance  $Z_L$ . This is a reasonable approximation for single-stage amplifiers of the type used in 8051-type devices. Then the gain of the amplifier as a function of frequency is

$$A = \frac{A_v Z_L}{Z_L + R_0}$$

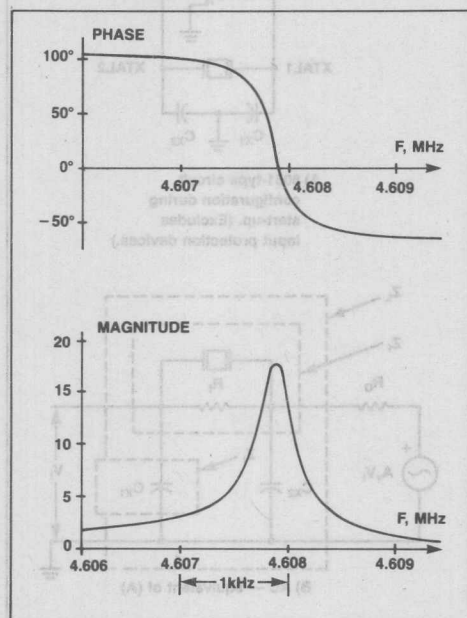


Figure 14 — Loop Gain versus Frequency (4.608 MHz Crystal)

The gain of the feedback network is

$$\beta = \frac{Z_1}{Z_1 + Z_2}$$

And the loop gain is

$$\beta A = \frac{1}{Z_1 + Z_2} \cdot \frac{A_v Z_L}{Z_L + R_0}$$

The impedances  $Z_L$ ,  $Z_1$ , and  $Z_2$  are defined in Figure 13B.

Figure 14 shows the way the loop gain thus calculated (using typical 8051-type parameters and a 4.608 MHz crystal) varies with frequency. The frequency of interest is the one for which the phase of the loop gain is zero. The accepted criterion for start-up is that the magnitude of the loop gain must exceed unity at this frequency. This is the frequency at which the circuit is in resonance. It corresponds very closely with the antiresonant frequency of the motional arm of the crystal in parallel with  $C_L$ .

Figure 15 shows the way the loop gain varies with frequency when the parameters of a 3.58MHz ceramic resonator are used in place of a crystal (the amplifier parameters being typical 8051, as in Figure 14). Note the different frequency scales.

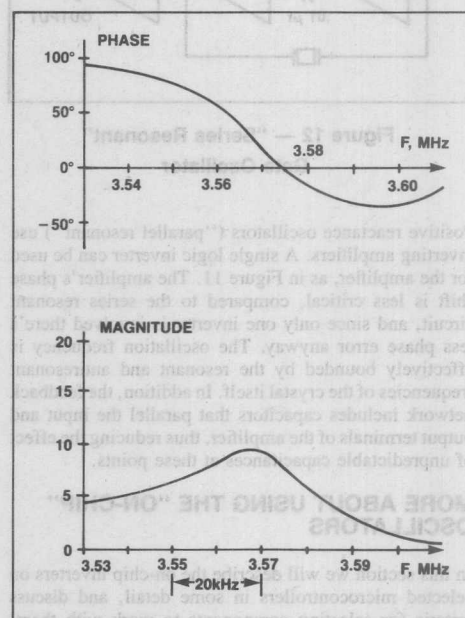


Figure 15 — Loop Gain versus Frequency (3.58MHz Ceramic)

### Start-Up Characteristics

It is common, in studies of feedback systems, to examine the behavior of the closed loop gain as a function of complex frequency  $s = \sigma + j\omega$ ; specifically, to determine the location of its poles in the complex plane. A pole is a point on the complex plane where the gain function goes to infinity. Knowledge of its location can be used to predict the response of the system to an input disturbance.

The way that the response function depends on the location of the poles is shown in Figure 16. Poles in the left half plane cause the response function to take the form of a damped sinusoid. Poles in the right half plane cause the response function to take the form of an exponentially growing sinusoid. In general,

$$v(t) \sim e^{at} \sin(\omega t + \theta)$$

where  $a$  is the real part of the pole frequency. Thus if the pole is in the right half plane,  $a$  is positive and the sinusoid grows. If the pole is in the left half plane,  $a$  is negative and the sinusoid is damped.

The same type of analysis can usefully be applied to oscillators. In this case, however, rather than trying to ensure that the poles are in the left half plane, we would seek to ensure that they're in the *right* half plane. An

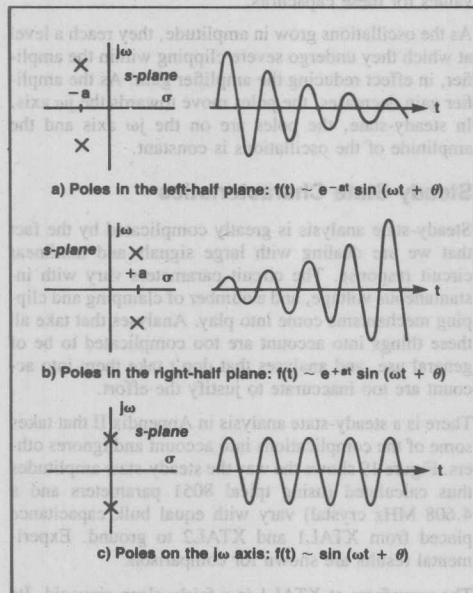


Figure 16 — Do You Know Where Your Poles are Tonight?

exponentially growing sinusoid is exactly what is wanted from an oscillator that has just been powered up.

The gain function of interest in oscillators is  $1/(1 - \beta A)$ . Its poles are at the complex frequencies where  $\beta A = 1 \angle 0^\circ$ , because that value of  $\beta A$  causes the gain function to go to infinity. The oscillator will start up if the real part of the pole frequency is positive. More importantly, the *rate* at which it starts up is indicated by how *much* greater than 0 the real part of the pole frequency is.

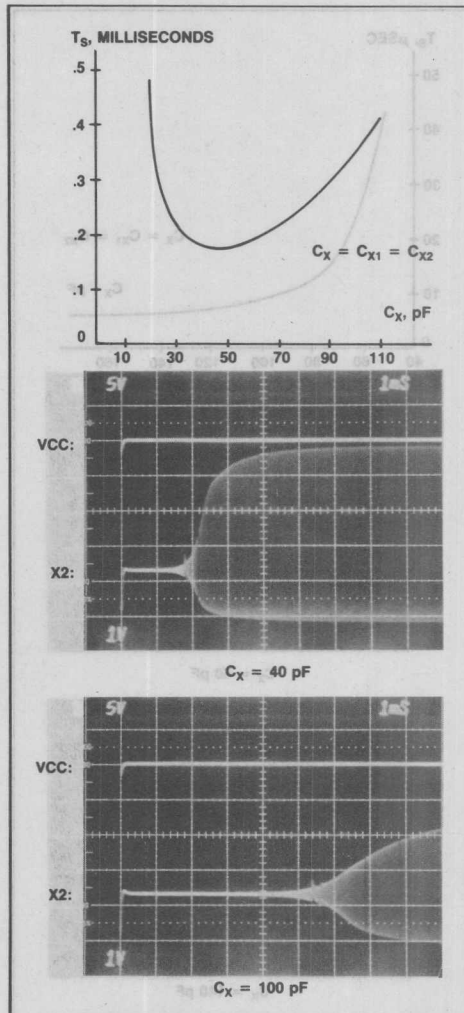


Figure 17 — Oscillator Start-Up (4.608 MHz Crystal from Standard Crystal Corp.)

The circuit in Figure 13B can be used to find the pole frequencies of the oscillator gain function. All that needs to be done is evaluate the impedances at complex frequencies  $\sigma + j\omega$  rather than just at  $\omega$ , and find the value of  $\sigma + j\omega$  for which  $\beta A = 1/\angle 0^\circ$ . The larger that value of  $\sigma$  is, the faster the oscillator will start up.

Of course, other things besides pole frequencies, things like the VCC rise time, are at work in determining the start-up time. But to the extent that the pole frequencies

do affect start-up time, we can obtain results like those in Figures 17 and 18.

To obtain these figures, the pole frequencies were computed for various values of capacitance  $C_X$  from XTAL1 and XTAL2 to ground (thus  $C_{X1} = C_{X2} = C_X$ ). Then a "time constant" for start-up was calculated as  $T_s = \frac{1}{\sigma}$  where  $\sigma$  is the real part of the pole frequency (rad/sec), and this time constant is plotted versus  $C_X$ .

A short time constant means faster start-up. A long time constant means slow start-up. Observations of actual start-ups are shown in the figures. Figure 17 is for a typical 8051 with a 4.608 MHz crystal, supplied by Standard Crystal Corp., and Figure 18 is for a typical 8051 with a 3.58MHz ceramic resonator supplied by NTK Technical Ceramics, Ltd.

It can be seen in Figure 17 that, for this crystal, values of  $C_X$  between 30 and 50 pF minimize start-up time, but that the exact value in this range is not particularly important, even if the start-up time itself is critical.

As previously mentioned, start-up time can be taken as an indication of start-up reliability. Start-up problems are normally associated with  $C_{X1}$  and  $C_{X2}$  being too small or too large for a given resonator. If the parameters of the resonator are known, curves such as in Figure 17 or 18 can be generated to define acceptable ranges of values for these capacitors.

As the oscillations grow in amplitude, they reach a level at which they undergo severe clipping within the amplifier, in effect reducing the amplifier gain. As the amplifier gain decreases, the poles move towards the  $j\omega$  axis. In steady-state, the poles are on the  $j\omega$  axis and the amplitude of the oscillations is constant.

### Steady-State Characteristics

Steady-state analysis is greatly complicated by the fact that we are dealing with large signals and nonlinear circuit response. The circuit parameters vary with instantaneous voltage, and a number of clamping and clipping mechanisms come into play. Analyses that take all these things into account are too complicated to be of general use, and analyses that don't take them into account are too inaccurate to justify the effort.

There is a steady-state analysis in Appendix II that takes some of the complications into account and ignores others. Figure 19 shows the way the steady-state amplitudes thus calculated (using typical 8051 parameters and a 4.608 MHz crystal) vary with equal bulk capacitance placed from XTAL1 and XTAL2 to ground. Experimental results are shown for comparison.

The waveform at XTAL1 is a fairly clean sinusoid. Its negative peak is normally somewhat below zero, at a level which is determined mainly by the input protection circuitry at XTAL1.

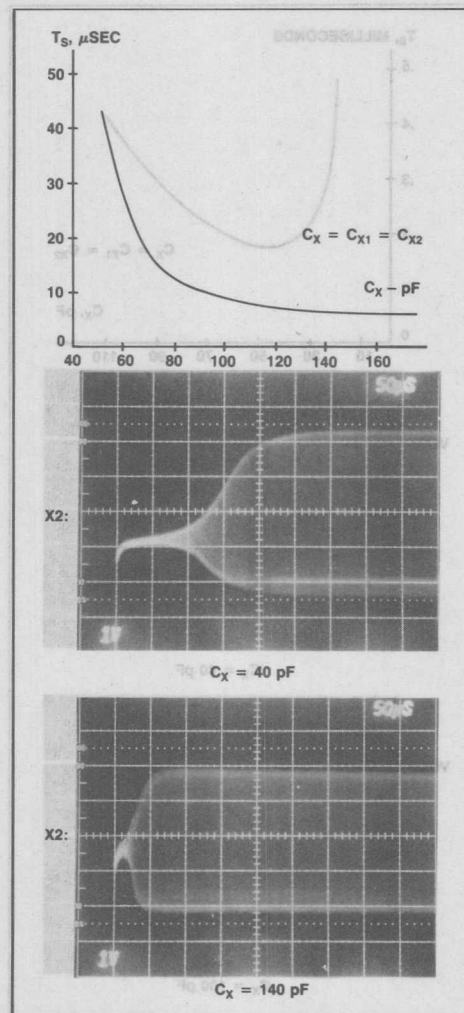
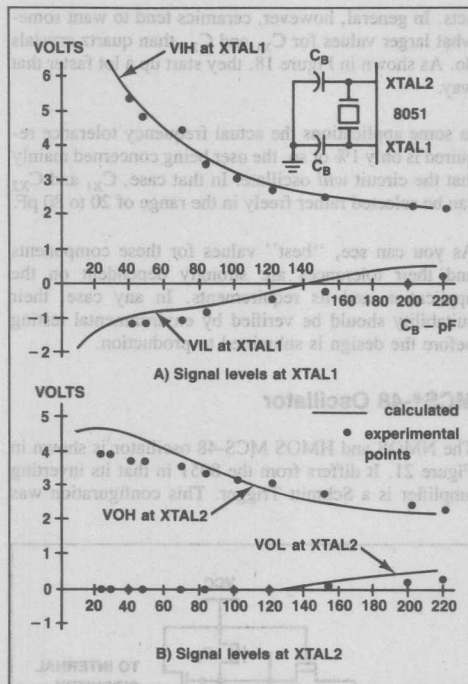


Figure 18 — Oscillator Start-Up (3.58 MHz Ceramic Resonator from NTK Technical Ceramics.)





**Figure 19 — Calculated and Experimental Steady-State Amplitudes vs. Bulk Capacitance from XTAL1 and XTAL2 to ground.**

The input protection circuitry consists of an ohmic resistor and an enhancement-mode FET with the gate and source connected to ground (VSS), as shown in Figure 20 for the 8051, and in Figure 21 for the 8048. Its function is to limit the positive voltage at the gate of the input FET to the avalanche voltage of the drain junction. If the input pin is driven below VSS, the drain and source of the protection FET interchange roles, so its gate is connected to what is now the drain. In this condition the device resembles a diode with the anode connected to VSS.

There is a parasitic pn junction between the ohmic resistor and the substrate. In the ROM parts (8051, 8048, etc.) the substrate is held at approximately  $-3V$  by the on-chip back-bias generator. In the EPROM parts (8751, 8748, etc.) the substrate is connected to VSS.

The effect of the input protection circuitry on the oscillator is that if the XTAL1 signal goes negative, its negative peak is clamped to  $-V_{DS}$  of the protection FET in the ROM parts, and to about  $-1.5V$  in the EPROM parts. These negative voltages on XTAL1 are in this application self-limiting and nondestructive!

The clamping action does, however, raise the DC level at XTAL1, which in turn tends to reduce the positive peak at XTAL2. The waveform at XTAL2 resembles a sinusoid riding on a DC level, and whose negative peaks are clipped off at zero.

Since it's normally the XTAL2 signal that drives the internal clocking circuitry, the question naturally arises as to how large this signal must be to reliably do its job. In fact, the XTAL2 signal doesn't have to meet the same VIH and VIL specifications that an external driver would have to. That's because as long as the oscillator is working, the on-chip amplifier is driving itself through its own 0-to-1 transition region, which is very nearly the same as the 0-to-1 transition region in the internal buffer that follows the oscillator. If some processing variations move the transition level higher or lower, the on-chip amplifier tends to compensate for it by the fact that its own transition level is correspondingly higher or lower. (In the 8096, it's the XTAL1 signal that drives the internal clocking circuitry, but the same concept applies.)

The main concern about the XTAL2 signal amplitude is as an indication of the general health of the oscillator. An amplitude of less than about 2.5V peak-to-peak indicates that start-up problems could develop in some units (with low gain) with some crystals (with high  $R_1$ ). The remedy is to either adjust the values of  $C_{X1}$  and/or  $C_{X2}$  or use a crystal with a lower  $R_1$ .

The amplitudes at XTAL1 and XTAL2 can be adjusted by changing the ratio of the capacitors from XTAL1 and XTAL2 to ground. Increasing the XTAL2 capacitance, for example, decreases the amplitude at XTAL2 and increases the amplitude at XTAL1 by about the same amount. Decreasing both caps increases both amplitudes.

### Pin Capacitance

Internal pin-to-ground and pin-to-pin capacitances at XTAL1 and XTAL2 will have some effect on the oscillator. These capacitances are normally taken to be in the range of 5 to 10 pF, but they are extremely difficult to evaluate. Any measurement of one such capacitance will necessarily include effects from the others. One advantage of the positive reactance oscillator is that the pin-to-ground capacitances are paralleled by external bulk capacitors, so a precise determination of their value is unnecessary. We would suggest that there is little justification for more precision than to assign them a value of 7 pF (XTAL1-to-ground and XTAL1-to-XTAL2). This value is probably not in error by more than 3 or 4 pF.

The XTAL2-to-ground capacitance is not entirely "pin capacitance," but more like an "equivalent output capacitance" of some 25 to 30 pF, having to include the effect of internal phase delays. This value will vary to some extent with temperature, processing, and frequency.



### MCS<sup>®</sup>-51 Oscillator

The on-chip amplifier on the HMOS MCS-51 family is shown in Figure 20. The drain load and feedback "resistors" are seen to be field-effect transistors. The drain load FET,  $R_D$ , is typically equivalent to about 1k to 3k-ohms. As an amplifier, the low frequency voltage gain is normally between -10 and -20, and the output resistance is effectively  $R_D$ .

The 8051 oscillator is normally used with equal bulk capacitors placed externally from XTAL1 to ground and from XTAL2 to ground. To determine a reasonable value of capacitance to use in these positions, given a crystal or ceramic resonator of known parameters, one can use the BASIC analysis in Appendix II to generate curves such as in Figures 17 and 18. This procedure will define a range of values that will minimize start-up time. We don't suggest that smaller values be used than those which minimize start-up time. Larger values than those can be used in applications where increased frequency stability is desired, at some sacrifice in start-up time.

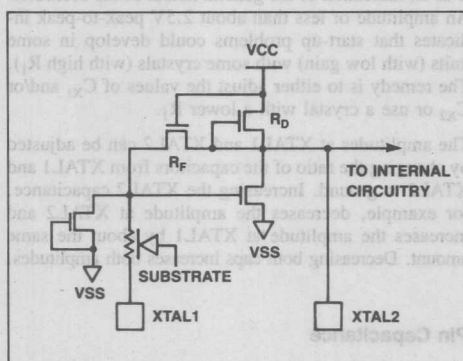


Figure 20 — MCS<sup>®</sup>-51 Oscillator Amplifier

Standard Crystal Corp. (reference 8) studied the use of their crystals with the MCS-51 family using skew samples supplied by Intel. They suggest putting 30 pF capacitors from XTAL1 and XTAL2 to ground, if the crystal is specified as described in reference 8. They noted that in that configuration and with crystals thus specified, the frequency accuracy was  $\pm 0.01\%$  and the frequency stability was  $\pm 0.005\%$ , and that a frequency accuracy of  $\pm 0.005\%$  could be obtained by substituting a 25 pF fixed cap in parallel with a 5-20 pF trimmer for one of the 30 pF caps.

MCS-51 skew samples have also been supplied to a number of ceramic resonator manufacturers for characterization with their products. These companies should be contacted for application information on their prod-

ucts. In general, however, ceramics tend to want somewhat larger values for  $C_{X1}$  and  $C_{X2}$  than quartz crystals do. As shown in Figure 18, they start up a lot faster that way.

In some applications the actual frequency tolerance required is only 1% or so, the user being concerned mainly that the circuit will oscillate. In that case,  $C_{X1}$  and  $C_{X2}$  can be selected rather freely in the range of 20 to 80 pF.

As you can see, "best" values for these components and their tolerances are strongly dependent on the application and its requirements. In any case, their suitability should be verified by environmental testing before the design is submitted to production.

### MCS<sup>®</sup>-48 Oscillator

The NMOS and HMOS MCS-48 oscillator is shown in Figure 21. It differs from the 8051 in that its inverting amplifier is a Schmitt Trigger. This configuration was

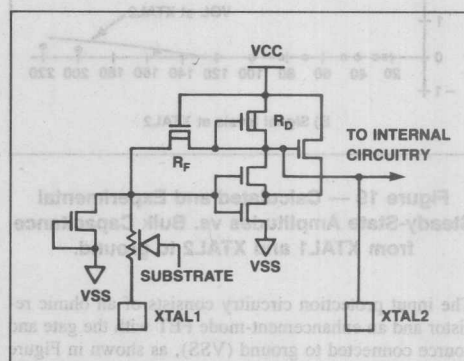


Figure 21 — MCS<sup>®</sup>-48 Oscillator Amplifier

chosen to prevent crosstalk from the TO pin, which is adjacent to the XTAL1 pin.

All Schmitt Trigger circuits exhibit a hysteresis effect, as shown in Figure 22. The hysteresis is what makes it less sensitive to noise. The same hysteresis allows any Schmitt Trigger to be used as a relaxation oscillator. All you have to do is connect a resistor from output to input, and a capacitor from input to ground, and the circuit oscillates in a relaxation mode as follows.

If the Schmitt Trigger output is at a logic high, the capacitor commences charging through the feedback resistor. When the capacitor voltage reaches the upper trigger point (UTP), the Schmitt Trigger output switches to a logic low and the capacitor commences discharging

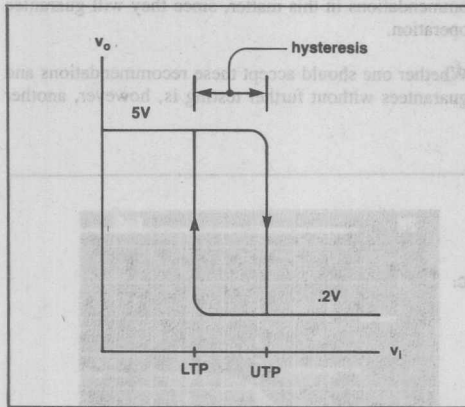


Figure 22 — Schmitt Trigger Characteristic

through the same resistor. When the capacitor voltage reaches the lower trigger point (LTP), the Schmitt Trigger output switches to a logic high again, and the sequence repeats. The oscillation frequency is determined by the RC time constant and the hysteresis voltage, UTP-LTP.

The 8048 can oscillate in this mode. It has an internal feedback resistor. All that's needed is an external capacitor from XTAL1 to ground. In fact, if a smaller external feedback resistor is added, an 8048 system could be designed to run in this mode. *Do it at your own risk!* This mode of operation is not tested, specified, documented, or encouraged in any way by Intel for the 8048. Future steppings of the device might have a different type of inverting amplifier (one more like the 8051). The CHMOS members of the MCS-48 family do not use a Schmitt Trigger as the inverting amplifier.

Relaxation oscillations in the 8048 must be avoided, and this is the major objective in selecting the off-chip components needed to complete the oscillator circuit.

When an 8048 is powered up, if VCC has a short rise time, the relaxation mode starts first. The frequency is normally about 50kHz. The resonator mode builds more slowly, but it eventually takes over and dominates the operation of the circuit. This is shown in Figure 23A.

Due to processing variations, some units seem to have a harder time coming out of the relaxation mode, particularly at low temperatures. In some cases the resonator oscillations may fail entirely, and leave the device in the relaxation mode. Most units will stick in the relaxation mode at any temperature if  $C_{X1}$  is larger than about 50 pF. Therefore,  $C_{X1}$  should be chosen with some care, particularly if the system must operate at lower temperatures.

One method that has proven effective in all units to  $-40^{\circ}\text{C}$  is to put 5 pF from XTAL1 to ground and 20 pF from XTAL2 to ground. Unfortunately, while this method does discourage the relaxation mode, it is not an optimal choice for the resonator mode. For one thing, it does not swamp the pin capacitance. Also, it makes for a rather high signal level at XTAL1 (8 or 9 volts peak-to-peak).

The question arises as to whether that level of signal at XTAL1 might damage the chip. Not to worry. The negative peaks are self-limiting and nondestructive. The positive peaks could conceivably damage the oxide, but in fact, NMOS chips (eg, 8048) and HMOS chips (eg, 8048H) are tested to a much higher voltage than that. The technology trend, of course, is to thinner oxides, as the devices shrink in size. For an extra margin of safety, the HMOS II chips (eg, 8048AH) have an internal diode clamp at XTAL1 to VCC.

In reality,  $C_{X1}$  doesn't have to be quite so small to avoid relaxation oscillations, if the minimum operating temperature is not  $-40^{\circ}\text{C}$ . For less severe temperature requirements, values of capacitance selected in much the same way as for an 8051 can be used. The circuit should be tested, however, at the system's lowest temperature limit.

Additional security against relaxation oscillations can be obtained by putting a 1M-ohm (or larger) resistor from XTAL1 to VCC. Pulling up the XTAL1 pin this way seems to discourage relaxation oscillations as effectively as any other method (Figure 23B).

Another thing that discourages relaxation oscillations is low VCC. The resonator mode, on the other hand, is much less sensitive to VCC. Thus if VCC comes up relatively slowly (several milliseconds rise time), the resonator mode is normally up and running before the relaxation mode starts (in fact, before VCC has even reached operating specs). This is shown in Figure 23C.

A secondary effect of the hysteresis is a shift in the oscillation frequency. At low frequencies, the output signal from an inverter without hysteresis leads (or lags) the input by 180 degrees. The hysteresis in a Schmitt Trigger, however, causes the output to lead the input by less than 180 degrees (or lag by more than 180 degrees), by an amount that depends on the signal amplitude, as shown in Figure 24. At higher frequencies, there are additional phase shifts due to the various reactances in the circuit, but the phase shift due to the hysteresis is still present. Since the total phase shift in the oscillator's loop gain is necessarily 0 or 360 degrees, it is apparent that as the oscillations build up, the frequency has to change to allow the reactances to compensate for the hysteresis. In normal operation, this additional phase shift due to hysteresis does not exceed a few degrees, and the resulting frequency shift is negligible.

Kyocera, a ceramic resonator manufacturer, studied the

use of some of their resonators (at 6.0MHz, 8.0MHz, and 11.0MHz) with the 8049H. Their conclusion as to the value of capacitance to use at XTAL1 and XTAL2 was that 33 pF is appropriate at all three frequencies. One should probably follow the manufacturer's rec-

ommendations in this matter, since they will guarantee operation.

Whether one should accept these recommendations and guarantees without further testing is, however, another

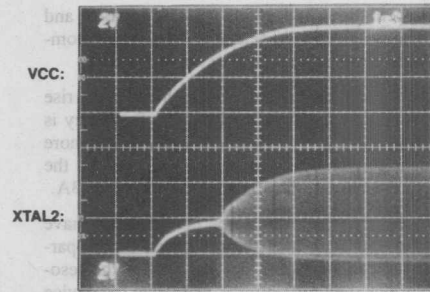
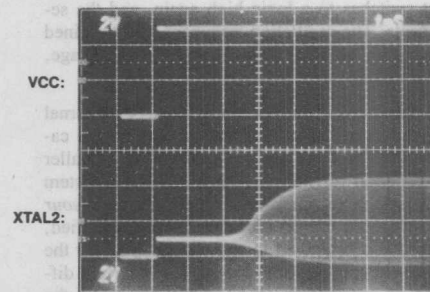
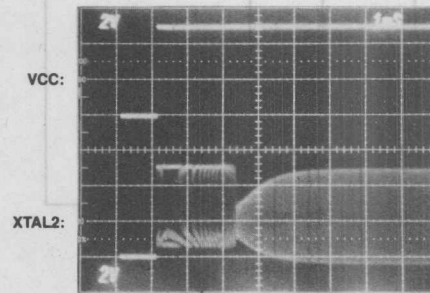
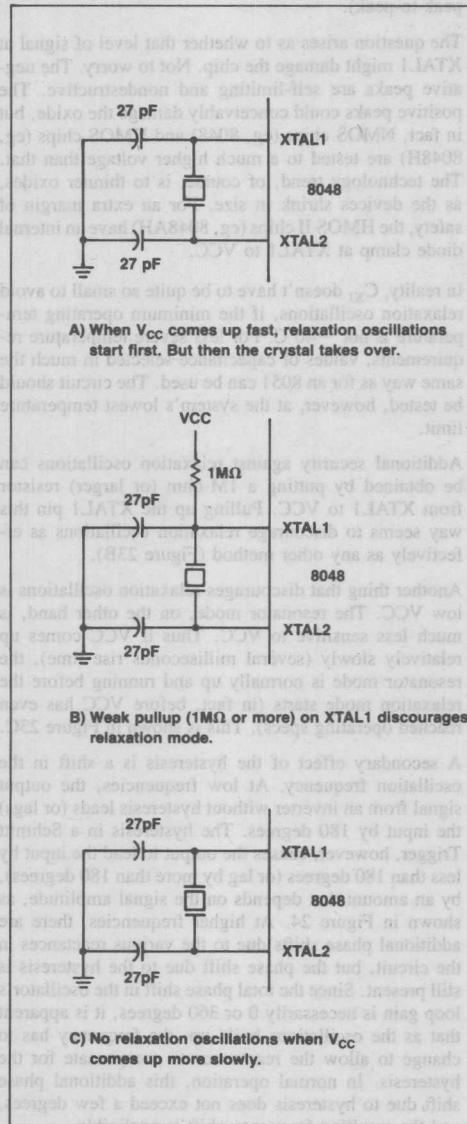


Figure 23 — Relaxation Oscillations in the 8048

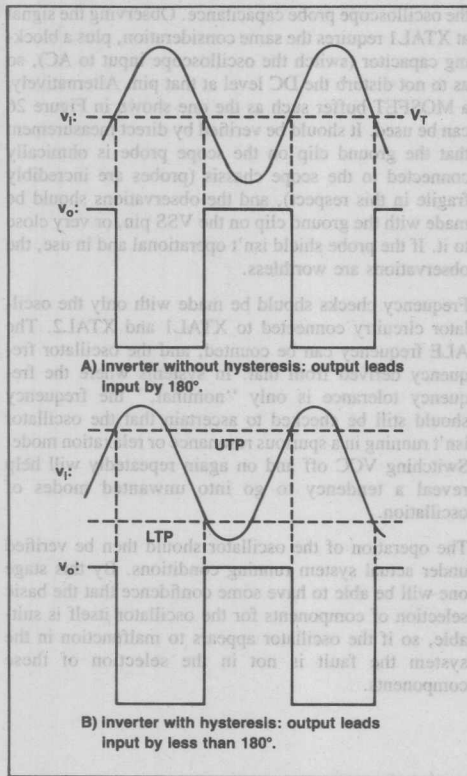


Figure 24 — Amplitude — Dependent Phase Shift in Schmitt Trigger

The first test is the actual application there may be significant differences in size capacitance, particularly in the case of the ceramic resonator. Not all users have found the recommendations to be without occasional problems. If you run into difficulties using their recommendations, both Intel and the ceramic resonator manufacturer want to know about it. It is to their interest, and ours, that such problems be resolved.

### Preproduction Tests

An oscillator design should never be considered ready for production until it has proven its ability to function acceptably well under worst-case environmental conditions and with parameters at their worst-case tolerance limits. Unexpected temperature effects in parts that may already be near their tolerance limits can prevent start-up of an oscillator that works perfectly well on the bench. For example, designers often overlook temperature effects in ceramic capacitors. (Some ceramics are down to 50% of their room-temperature values at  $-20^{\circ}\text{C}$  and  $+60^{\circ}\text{C}$ .) The problem here isn't just one of

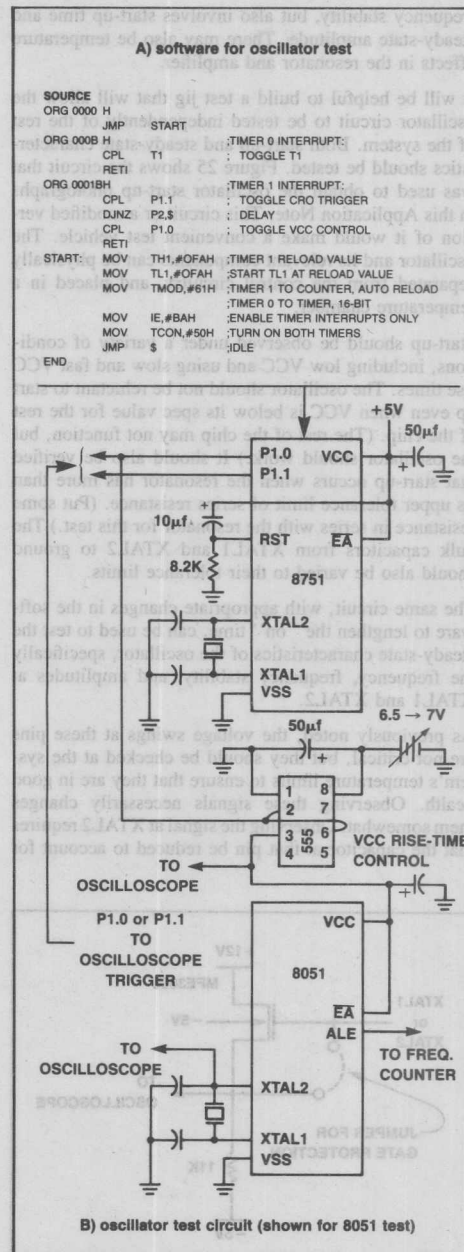


Figure 25 — Oscillator Test Circuit and Software



frequency stability, but also involves start-up time and steady-state amplitude. There may also be temperature effects in the resonator and amplifier.

It will be helpful to build a test jig that will allow the oscillator circuit to be tested independently of the rest of the system. Both start-up and steady-state characteristics should be tested. Figure 25 shows the circuit that was used to obtain the oscillator start-up photographs in this Application Note. This circuit or a modified version of it would make a convenient test vehicle. The oscillator and its relevant components can be physically separated from the control circuitry, and placed in a temperature chamber.

Start-up should be observed under a variety of conditions, including low VCC and using slow and fast VCC rise times. The oscillator should not be reluctant to start up even when VCC is below its spec value for the rest of the chip. (The rest of the chip may not function, but the oscillator should work.) It should also be verified that start-up occurs when the resonator has more than its upper tolerance limit of series resistance. (Put some resistance in series with the resonator for this test.) The bulk capacitors from XTAL1 and XTAL2 to ground should also be varied to their tolerance limits.

The same circuit, with appropriate changes in the software to lengthen the "on" time, can be used to test the steady-state characteristics of the oscillator, specifically the frequency, frequency stability, and amplitudes at XTAL1 and XTAL2.

As previously noted, the voltage swings at these pins are not critical, but they should be checked at the system's temperature limits to ensure that they are in good health. Observing these signals necessarily changes them somewhat. Observing the signal at XTAL2 requires that the capacitor at that pin be reduced to account for

the oscilloscope probe capacitance. Observing the signal at XTAL1 requires the same consideration, plus a blocking capacitor (switch the oscilloscope input to AC), so as to not disturb the DC level at that pin. Alternatively, a MOSFET buffer such as the one shown in Figure 26 can be used. It should be verified by direct measurement that the ground clip on the scope probe is ohmically connected to the scope chassis (probes are incredibly fragile in this respect), and the observations should be made with the ground clip on the VSS pin, or very close to it. If the probe shield isn't operational and in use, the observations are worthless.

Frequency checks should be made with only the oscillator circuitry connected to XTAL1 and XTAL2. The ALE frequency can be counted, and the oscillator frequency derived from that. In systems where the frequency tolerance is only "nominal," the frequency should still be checked to ascertain that the oscillator isn't running in a spurious resonance or relaxation mode. Switching VCC off and on again repeatedly will help reveal a tendency to go into unwanted modes of oscillation.

The operation of the oscillator should then be verified under actual system running conditions. By this stage one will be able to have some confidence that the basic selection of components for the oscillator itself is suitable, so if the oscillator appears to malfunction in the system the fault is not in the selection of these components.

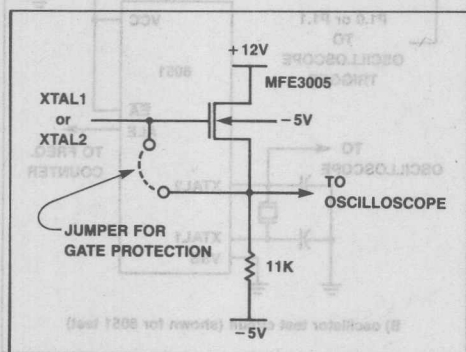


Figure 26 — MOSFET Buffer for Observing Oscillator Signals

### Troubleshooting Oscillator Problems

The first thing to consider in case of difficulty is that between the test jig and the actual application there may be significant differences in stray capacitances, particularly if the actual application is on a multi-layer board.

Noise glitches, that aren't present in the test jig but are in the application board, are another possibility. Capacitive coupling between the oscillator circuitry and other signals has already been mentioned as a source of miscounts in the internal clocking circuitry. Inductive coupling is also possible, if there are strong currents nearby. These problems are a function of the PCB layout.

Surrounding the oscillator components with "quiet" traces (VCC and ground, for example) will alleviate capacitive coupling to signals that have fast transition times. To minimize inductive coupling, the PCB layout should minimize the areas of the loops formed by the oscillator components. These are the loops that should be checked:

- XTAL1 through the resonator to XTAL2;
- XTAL1 through CX1 to the VSS pin;
- XTAL2 through CX2 to the VSS pin.



It is not unusual to find that the grounded ends of  $C_{X1}$  and  $C_{X2}$  eventually connect up to the VSS pin only after looping around the farthest ends of the board. Not good.

Finally, it should not be overlooked that software problems sometimes imitate the symptoms of a slow-starting oscillator or incorrect frequency. Never underestimate the perversity of a software problem.

### REFERENCES

1. Frerking, M. E., *Crystal Oscillator Design and Temperature Compensation*, Van Nostrand Reinhold, 1978.
2. Bottom, V., "The Crystal Unit as a Circuit Component," Ch. 7, *Introduction to Quartz Crystal Unit Design*, Van Nostrand Reinhold, 1982.
3. Parzen, B., *Design of Crystal and Other Harmonic Oscillators*, John Wiley & Sons, 1983.
4. Holmbeck, J. D., "Frequency Tolerance Limitations with Logic Gate Clock Oscillators," *31st Annual Frequency Control Symposium*, June, 1977.
5. Roberge, J. K., "Nonlinear Systems," Ch. 6, *Operational Amplifiers: Theory and Practice*, Wiley, 1975.
6. Eaton, S. S., *Timekeeping Advances Through COS/MOS Technology*, RCA Application Note ICAN-6086.
7. Eaton, S. S., *Micropower Crystal-Controlled Oscillator Design Using RCA COS/MOS Inverters*, RCA Application Note ICAN-6539.
8. Fisher, J. B., *Crystal Specifications for the Intel 8031/8051/8751 Microcontrollers*, Standard Crystal Corp. Design Data Note #2F.
9. Murata Mfg. Co., Ltd., *Ceramic Resonator "Ceralock" Application Manual*.
10. Kyoto Ceramic Co., Ltd., *Adaptability Test Between Intel 8049H and Kyocera Ceramic Resonators*.
11. Kyoto Ceramic Co., Ltd., *Technical Data on Ceramic Resonator Model KBR-6.0M, KBR-8.0M, KBR-11.0M Application for 8051 (Intel)*.
12. NTK Technical Ceramic Division, NGK Spark Plug Co., Ltd., *NTKK Ceramic Resonator Manual*.

It is not unusual to find that the grounded ends of  $C_{11}$  and  $C_{22}$  eventually connect up to the VSS pin only after looking around the farthest ends of the board. Not good.

Finally, it should not be overlooked that software problems sometimes imitate the symptoms of a slow-starting oscillator or incorrect frequency. Never underestimate the perversity of a software problem.

## REFERENCES

1. Fackling, M. E., "Crystal Oscillator Design and Temperature Compensation," Van Nostrand Reinhold, 1978.
2. Bottom, V., "The Crystal as a Circuit Component," Ch. 7, Introduction and Unit Design, Van Nostrand Reinhold, 1983.
3. Pavesi, B., "Design of Crystal and Other Harmonic Oscillators," John Wiley & Sons, 1983.
4. Holmbeck, J. D., "Frequency Tolerance Limitations with Logic Gate Oscillators," 31st Annual Frequency Control Symposium, June 1977.
5. Roberts, J. K., "Nonlinear Systems," Ch. 6, Operational Amplifiers: Theory and Practice, Wiley, 1975.
6. Eaton, S. S., "Timekeeping Advances Through CMOS Technology," RCA Application Note ICAN-6086.
7. Eaton, S. S., "Micropower Crystal-Controlled Oscillator Design Using RCA CMOS Inverters," RCA Application Note ICAN-6239.
8. "Crystal Specifications for the Microcontroller," Standard Data Note #2F.
9. Murata Mfg. Co., Ltd., "Ceramic Resonator," "Cerulock," Application Manual.
10. Kyoto Ceramic Co., Ltd., "Adaptability Test Between Inlet 8049H and Kyocera Ceramic Resonators."
11. Kyoto Ceramic Co., Ltd., "Technical Data on Ceramic Resonator Model KBR-8.0M, KBR-8.0M, KBR-11.0M Application for 8051 (Intel)."
12. NTK Technical Ceramic Division, NGK Spark Plug Co., Ltd., "NTK Ceramic Resonator Manual."

APPENDIX I ..... 22  
APPENDIX II ..... 24

## APPENDIX I

# QUARTZ AND CERAMIC RESONATOR FORMULAS

Based on the equivalent circuit of the crystal, the impedance of the crystal is

$$Z_{XTAL} = \frac{(R_1 + j\omega L_1 + 1/j\omega C_1)(1/j\omega C_0)}{R_1 + j\omega L_1 + 1/j\omega C_1 + 1/j\omega C_0}$$

After some algebraic manipulation, this calculation can be written in the form

$$Z_{XTAL} = \frac{1}{j\omega(C_1 + C_0)} \cdot \frac{1 - \omega^2 L_1 C_1 + j\omega R_1 C_1}{1 - \omega^2 L_1 C_T + j\omega R_1 C_T}$$

where  $C_T$  is the capacitance of  $C_1$  in series with  $C_0$ :

$$C_T = \frac{C_1 C_0}{C_1 + C_0}$$

The impedance of the crystal in parallel with an external load capacitance  $C_L$  is the same expression, but with  $C_0 + C_L$  substituted for  $C_0$ :

$$Z_{XTAL \parallel CL} = \frac{1}{j\omega(C_1 + C_0 + C_L)} \cdot \frac{1 - \omega^2 L_1 C_1 + j\omega R_1 C_1}{1 - \omega^2 L_1 C_T' + j\omega R_1 C_T'}$$

where  $C_T'$  is the capacitance of  $C_1$  in series with  $(C_0 + C_L)$ :

$$C_T' = \frac{C_1(C_0 + C_L)}{C_1 + C_0 + C_L}$$

The impedance of the crystal in series with the load capacitance is

$$\begin{aligned} Z_{XTAL + CL} &= Z_{XTAL} + \frac{1}{j\omega C_L} \\ &= \frac{C_L + C_1 + C_0}{j\omega C_L(C_1 + C_0)} \cdot \frac{1 - \omega^2 L_1 C_T' + j\omega R_1 C_T'}{1 - \omega^2 L_1 C_T + j\omega R_1 C_T} \end{aligned}$$

where  $C_T$  and  $C_T'$  are as defined above.

The phase angles of these impedances are readily obtained from the impedance expressions themselves:

$$\begin{aligned} \theta_{XTAL} &= \arctan \frac{\omega R_1 C_1}{1 - \omega^2 L_1 C_1} \\ &\quad - \arctan \frac{\omega R_1 C_T}{1 - \omega^2 L_1 C_T} - \frac{\pi}{2} \\ \theta_{XTAL \parallel CL} &= \arctan \frac{\omega R_1 C_1}{1 - \omega^2 L_1 C_1} \\ &\quad - \arctan \frac{\omega R_1 C_T'}{1 - \omega^2 L_1 C_T'} - \frac{\pi}{2} \end{aligned}$$

$$\theta_{XTAL + CL} = \arctan \frac{\omega R_1 C_T'}{1 - \omega^2 L_1 C_T'} - \arctan \frac{\omega R_1 C_T}{1 - \omega^2 L_1 C_T} - \frac{\pi}{2}$$

The resonant ("series resonant") frequency is the frequency at which the phase angle is zero and the impedance is low. The antiresonant ("parallel resonant") frequency is the frequency at which the phase angle is zero and the impedance is high.

Each of the above  $\theta$ -expressions contains two arctan functions. Setting the denominator of the argument of the first arctan function to zero gives (approximately) the "series resonant" frequency for that configuration. Setting the denominator of the argument of the second arctan function to zero gives (approximately) the "parallel resonant" frequency for that configuration.

For example, the resonant frequency of the crystal is the frequency at which

$$1 - \omega^2 L_1 C_1 = 0$$

Thus,

$$\omega_s = \frac{1}{\sqrt{L_1 C_1}}$$

or

$$f_s = \frac{1}{2\pi\sqrt{L_1 C_1}}$$

It will be noted that the series resonant frequency of the "XTAL + CL" configuration (crystal in series with CL) is the same as the parallel resonant frequency of the "XTAL  $\parallel$  CL" configuration (crystal in parallel with  $C_L$ ). This is the frequency at which

$$1 - \omega^2 L_1 C_T' = 0$$

Thus,

$$\omega_a = \frac{1}{\sqrt{L_1 C_T'}}$$

or

$$f_a = \frac{1}{2\pi\sqrt{L_1 C_T'}}$$

This fact is used by crystal manufacturers in the process of calibrating a crystal to a specified load capacitance.

By subtracting the resonant frequency of the crystal from its antiresonant frequency, one can calculate the range of frequencies over which the crystal reactance is positive:

$$\begin{aligned} f_a - f_s &= f_s (\sqrt{1 + C_1/C_0} - 1) \\ &= f_s \left( \frac{C_1}{2C_0} \right) \end{aligned}$$

Given typical values for  $C_1$  and  $C_0$ , this range can hardly exceed 0.5% of fs. Unless the inverting amplifier in the positive reactance oscillator is doing something very strange indeed, the oscillation frequency is bound to be accurate to that percentage whether the crystal was calibrated for series operation or to any unspecified load capacitance.

### Equivalent Series Resistance

ESR is the real part of  $Z_{XTAL}$  at the oscillation frequency. The oscillation frequency is the parallel resonant frequency of the "XTAL || CL" configuration (which is the same as the series resonant frequency of the "XTAL+CL" configuration). Substituting this frequency into the  $Z_{XTAL}$  expression yields, after some algebraic manipulation,

$$ESR = \frac{R_1 \left( \frac{C_0 + C_L}{C_L} \right)^2}{1 + \omega^2 C_1^2 \left( \frac{C_0 + C_L}{C_L} \right)^2}$$

$$\approx R_1 \left( 1 + \frac{C_0^2}{C_L^2} \right)$$

For example, the resonant frequency of the crystal is the frequency at which

$$\omega = \frac{1}{\sqrt{L_1 C_1}}$$

$$f = \frac{1}{2\pi \sqrt{L_1 C_1}}$$

It will be noted that the series resonant frequency of the "XTAL+CL" configuration (crystal in series with CL) is the same as the parallel resonant frequency of the "XTAL || CL" configuration (crystal in parallel with CL). This is the frequency at which

$$1 - \omega^2 L_1 C_1 = 0$$

$$\omega = \frac{1}{\sqrt{L_1 C_1}}$$

$$f = \frac{1}{2\pi \sqrt{L_1 C_1}}$$

This fact is used by crystal manufacturers in the process of calibrating a crystal to a specified load capacitance.

By substituting the resonant frequency of the crystal (from its manufacturer's frequency) one can calculate the range of frequencies over which the crystal reactance is positive:

$$f - f_0 = f \left( \sqrt{1 + \frac{C_0^2}{C_L^2}} - 1 \right)$$

$$f = \frac{f_0}{\left( \sqrt{1 + \frac{C_0^2}{C_L^2}} - 1 \right)}$$

### Drive Level

The power dissipated by the crystal is  $I_1^2 R_1$ , where  $I_1$  is the RMS current in the motional arm of the crystal. This current is given by  $V_x / |Z_1|$ , where  $V_x$  is the RMS voltage across the crystal, and  $|Z_1|$  is the magnitude of the impedance of the motional arm. At the oscillation frequency, the motional arm is a positive (inductive) reactance in parallel resonance with  $(C_0 + C_L)$ . Therefore  $|Z_1|$  is approximately equal to the magnitude of the reactance of  $(C_0 + C_L)$ :

$$|Z_1| = \frac{1}{2\pi f(C_0 + C_L)}$$

where  $f$  is the oscillation frequency. Then,

$$P = I_1^2 R_1 = \left( \frac{V_x}{|Z_1|} \right)^2 R_1$$

$$= [2\pi f(C_0 + C_L) V_x]^2 R_1$$

The waveform of the voltage across the crystal (XTAL1 to XTAL2) is approximately sinusoidal. If its peak value is  $V_{CC}$ , then  $V_x$  is  $V_{CC}/\sqrt{2}$ . Therefore,

$$P = 2R_1 [\pi f(C_0 + C_L) V_{CC}]^2$$

$$C_T = \frac{C_1(C_0 + C_L)}{C_1(C_0 + C_L) + C_L}$$

The impedance of the crystal in series with the load capacitance is

$$Z_{XTAL+CL} = \frac{1}{j\omega C_T + \frac{1}{j\omega C_1 + \frac{1}{j\omega L_1 + \frac{1}{j\omega C_0}}}}$$

$$= \frac{C_1 + C_0}{j\omega C_1(C_0 + C_L) + j\omega C_L} \cdot \frac{1 - \omega^2 L_1 C_1 + j\omega R_1 C_1}{1 - \omega^2 L_1(C_0 + C_L) + j\omega R_1(C_0 + C_L)}$$

where  $C_T$  and  $C_L$  are as defined above. The phase angles of these impedances are readily obtained from the impedance expressions themselves:

$$\theta_{XTAL} = \arctan \frac{\omega R_1 C_1}{1 - \omega^2 L_1 C_1}$$

$$-\arctan \frac{\omega R_1 C_L}{1 - \omega^2 L_1(C_0 + C_L)}$$

$$\theta_{XTAL+CL} = \arctan \frac{\omega R_1 C_1}{1 - \omega^2 L_1 C_1}$$

$$-\arctan \frac{\omega R_1 C_L}{1 - \omega^2 L_1(C_0 + C_L)}$$

## APPENDIX II

# OSCILLATOR ANALYSIS PROGRAM

The program is written in BASIC. BASIC is excruciatingly slow, but it has some advantages. For one thing, more people know BASIC than FORTRAN. In addition, a BASIC program is easy to develop, modify, and "fiddle around" with. Another important advantage is that a BASIC program can run on practically any small computer system.

Its slowness is a problem, however. For example, the routine which calculates the "start-up time constant" discussed in the text may take several hours to complete. A person who finds this program useful may prefer to convert it to FORTRAN, if the facilities are available.

### Limitations of the Program

The program was developed with specific reference to 8051-type oscillator circuitry. That means the on-chip amplifier is a simple inverter, and not a Schmitt Trigger. The 8096, the 80C51, the 80C48 and 80C49 all have simple inverters. The 8096 oscillator is almost identical to the 8051, differing mainly in the input protection circuitry. The CHMOS amplifiers have somewhat different parameters (higher gain, for example), and different transition levels than the 8051.

The MCS-48 family is specifically included in the program only to the extent that the input-output curve used in the steady-state analysis is that of a Schmitt Trigger, if the user identifies the device under analysis as an MCS-48 device. The analysis does not include the voltage dependent phase shift of the Schmitt Trigger.

The clamping action of the input protection circuitry is important in determining the steady-state amplitudes. The steady-state routine accounts for it by setting the negative peak of the XTAL1 signal at a level which depends on the amplitude of the XTAL1 signal in accordance with experimental observations. It's an exercise in curve-fitting. A user may find a different type of curve works better. Later steppings of the chips may behave differently in this respect, having somewhat different types of input protection circuitry.

It should be noted that the analysis ignores a number of important items, such as high-frequency effects in the on-chip circuitry. These effects are difficult to predict, and are no doubt dependent on temperature, frequency, and device sample. However, they can be simulated to a reasonable degree by adding an "output capacitance" of about 20 pF to the circuit model (ie, in parallel with CX2), as described below.

### Notes on Using the Program

The program asks the user to input values for various circuit parameters. First the crystal (or ceramic resonator) parameters are asked for. These are R1, L1, C1, and C0. The manufacturer can supply these values for selected samples. To obtain any kind of correlation between calculation and experiment, the values of these parameters must be known for the specific sample in the test circuit. The value that should be entered for C0 is the C0 of the crystal itself plus an estimated 7 pF to account for the XTAL1-to-XTAL2 pin capacitance, plus any other stray capacitance paralleling the crystal that the user may feel is significant enough to be included.

Then the program asks for the values of the XTAL1-to-ground and XTAL2-to-ground capacitances. For CXTAL1, enter the value of the externally connected bulk capacitor plus an estimated 7 pF for pin capacitance. For CXTAL2, enter the value of the externally connected bulk capacitor plus an estimated 7 pF for pin capacitance plus about 20 pF to simulate high-frequency roll-off and phase shifts in the on-chip circuitry.

Next the program asks for values for the small-signal parameters of the on-chip amplifier. Typically, for the 8051/8751,

Amplifier Gain Magnitude = 15  
Feedback Resistance = 2300k-ohms  
Output Resistance = 2k-ohms

The same values can be used for MCS-48 (NMOS and HMOS) devices, but they are difficult to verify, because the Schmitt Trigger does not lend itself to small-signal measurements.



APRIL 8, 1983

```

100 DEFDBL C,D,F,G,L,P,R,S,X
200 REM
300 REM *****
400 REM
500 REM
600 REM
700 REM
800 REM FNZM(R,X) = MAGNITUDE OF A COMPLEX NUMBER, |R+jX|
900 DEF FNZM(R,X) = SQR(R^2+X^2)
1000 REM
1100 REM FNZP(R,X) = ANGLE OF A COMPLEX NUMBER
1200 REM = 180/PI*ARCTAN(X/R)
1300 REM = 180/PI*ARCTAN(X/R) + 180 IF R<0 AND X<0
1400 REM = 180/PI*ARCTAN(X/R) - 180 IF R<0 AND X>0
1500 DEF FNZP(R,X) = 180/PI*ATN(X/R) - ((SGN(R)-1)*SGN(X)*90)
1600 REM
1700 REM INDUCTIVE IMPEDANCE AT COMPLEX FREQUENCY S+jF (HZ)
1800 REM Z = 2*PI*S*L + j2*PI*F*L
1900 REM = FNRL(S,L) + jFNXL(F,L)
2000 DEF FNRL(SL,LL) = 2*PI*SL*LL
2100 DEF FNXL(FL,LL) = 2*PI*FL*LL
2200 REM
2300 REM CAPACITIVE IMPEDANCE AT COMPLEX FREQUENCY S+jF (HZ)
2400 REM Z = 1/[2*PI*(S+jF)*C]
2500 REM = S/[2*PI*(S^2+F^2)*C] + j(-F)/[2*PI*(S^2+F^2)*C]
2600 REM = FNRC(S,F,C) + jFNXC(S,F,C)
2700 DEF FNRC(SC,FC,CC) = SC/(2*PI*(SC^2+FC^2)*CC)
2800 DEF FNXC(SC,FC,CC) = -FC/(2*PI*(SC^2+FC^2)*CC)
2900 REM
3000 REM RATIO OF TWO COMPLEX NUMBERS
3100 REM RA+jXA RA*RB+XA*XB XA*RB-RA*XB
3200 REM ----- + j -----
3300 REM RB+jXB RB^2+XB^2 RB^2+XB^2
3400 REM = FNRR(RA,XA,RB,XB) + jFNXR(RA,XA,RB,XB)
3500 DEF FNRR(RA,XA,RB,XB) = (RA*RB+XA*XB)/(RB^2+XB^2)
3600 DEF FNXR(RA,XA,RB,XB) = (XA*RB-RA*XB)/(RB^2+XB^2)
3700 REM
3800 REM PRODUCT OF TWO COMPLEX NUMBERS
3900 REM (RA+jXA)*(RB+jXB) = RA*RB-XA*XB + j(XA*RB+RA*XB)
4000 REM = FNRM(RA,XA,RB,XB) + jFNXM(RA,XA,RB,XB)
4100 DEF FNRM(RA,XA,RB,XB) = RA*RB - XA*XB
4200 DEF FNXM(RA,XA,RB,XB) = RA*XB + RB*XA
4300 REM
4400 REM
4500 REM PARALLEL IMPEDANCES
4600 REM (RA+jXA)*(RB+jXB)
4700 REM (RA+jXA):(RB+jXB) = -----
4800 REM RA*RB + j(XA*XB)
4900 REM
5000 REM RA*(RB^2+XB^2)+RB*(RA^2+XA^2) XA*(RB^2+XB^2)+XB*(RA^2+XA^2)
5100 REM = ----- + j -----
5200 REM (RA+RB)^2 + (XA+XB)^2 (RA+RB)^2 + (XA+XB)^2
5300 REM
5400 REM = FNRP(RA,XA,RB,XB) + jFNXP(RA,XA,RB,XB)
5500 DEF FNRP(RA,XA,RB,XB) = (RA*(RB^2+XB^2) + RB*(RA^2+XA^2))/(RA+RB)^2 + (XA+XB)^2
5600 DEF FNXP(RA,XA,RB,XB) = (XA*(RB^2+XB^2) + XB*(RA^2+XA^2))/(RA+RB)^2 + (XA+XB)^2
5700 REM
5800 REM *****
5900 REM
6000 REM BEGIN COMPUTATIONS
6100 REM
6200 LET PI = 3.141592654#
6300 REM
6400 REM DEFINE CIRCUIT PARAMETERS
6500 GOSUB 14500
6600 REM
6700 REM ESTABLISH NOMINAL RESONANT AND ANTIRESONANT CRYSTAL FREQUENCIES
6800 FS = FIX(1/(2*PI*SGR(L1*C1)))
6900 FA = FIX(1/(2*PI*SGR(L1*C1*CO/(C1+CO))))
7000 PRINT
7100 PRINT "XTAL IS SERIES RESONANT AT ";FS;" HZ"
7200 PRINT " PARALLEL RESONANT AT ";FA;" HZ"
7300 PRINT
7400 PRINT "SELECT: 1. LIST PARAMETERS"
7500 PRINT " 2. CIRCUIT ANALYSIS"
7600 PRINT " 3. OSCILLATION FREQUENCY"
7700 PRINT " 4. START-UP TIME CONSTANT"
7800 PRINT " 5. STEADY-STATE ANALYSIS"

```

```

7900 PRINT
8000 INPUT N
8100 IF N=1 THEN PRINT ELSE 8600
8200 REM
8300 REM ----- LIST PARAMETERS -----
8400 GOSUB 17100
8500 GOTO 6800
8600 IF N=2 THEN PRINT ELSE 9400
8700 REM
8800 REM ----- CIRCUIT ANALYSIS -----
8900 PRINT " FREQUENCY S+JF: TYPE (S), (F). "
9000 INPUT SQ,FQ
9100 GOSUB 20200
9200 GOSUB 26600
9300 GOTO 6800
9400 IF N=3 THEN 10300 ELSE 11000
9500 REM
9600 REM ----- OSCILLATION FREQUENCY -----
9700 CL = CX*CY/(CX+CY) + CO
9800 FQ = FIX(1/(2*PI*SQR(L1*C1*CL/(C1+CL))))
9900 SQ = 0
10000 DF = FIX(10*INT(LOG(FA-FS)/LOG(10)-2)+.5)
10100 DS = 0
10200 RETURN
10300 GOSUB 9700
10400 GOSUB 30300
10500 PRINT
10600 PRINT
10700 PRINT "FREQUENCY AT WHICH LOOP GAIN HAS ZERO PHASE ANGLE: "
10800 GOSUB 26600
10900 GOTO 6800
11000 IF N=4 THEN PRINT ELSE 12200
11100 REM
11200 REM ----- START-UP TIME CONSTANT -----
11300 PRINT "THIS WILL TAKE SOME TIME ...."
11400 GOSUB 9700
11500 GOSUB 37700
11600 PRINT
11700 PRINT
11800 PRINT "FREQUENCY AT WHICH LOOP GAIN = 1 AT 0 DEGREES: "
11900 GOSUB 26600
12000 PRINT : PRINT "THIS YIELDS A START-UP TIME CONSTANT OF ";CSNG(1000000/(2*PI*SQ));" MICROSECS"
12100 GOTO 6800
12200 IF N=5 THEN PRINT ELSE 7300
12300 REM
12400 REM ----- STEADY-STATE ANALYSIS -----
12500 PRINT "STEADY-STATE ANALYSIS"
12600 PRINT
12700 PRINT "SELECT: 1. 8031/8051"
12800 PRINT "                2. 8751"
12900 PRINT "                3. 8035/8039/8040/8048/8049"
13000 PRINT "                4. 8748/8749"
13100 INPUT ICX
13200 IF ICX<1 OR ICX>4 THEN 12600
13300 GOSUB 46900
13400 GOTO 7300
13500 REM SUBROUTINE BELOW DEFINES INPUT-OUTPUT CURVE OF OSCILLATOR CKT
13600 IF ICX>2 AND VO=5 AND VI<2 THEN RETURN
13700 VO = -10*VI + 15
13800 IF VO>5 THEN VO = 5
13900 IF VO<2 THEN VO = .2
14000 IF ICX>2 AND VO>2 THEN VO = 5
14100 RETURN
14200 REM
14300 REM *****
14400 REM
14500 REM
14600 REM
14700 INPUT " R1 (OHMS)";R1
14800 INPUT " L1 (HENRY)";L1
14900 INPUT " C1 (PF)";X
15000 C1 = X*1E-12
15100 INPUT " CO (PF)";X
15200 CO = X*1E-12
15300 INPUT " CXTAL1 (PF)";X
15400 CX = X*1E-12
15500 INPUT " CXTAL2 (PF)";X
15600 CY = X*1E-12

```

```

15700 INPUT " GAIN FACTOR MAGNITUDE";AV#
15800 INPUT " AMP FEEDBACK RESISTANCE (K-OHMS)";X
15900 RX = X*1000#
16000 INPUT " AMP OUTPUT RESISTANCE (K-OHMS)";X
16100 RO = X*1000#
16200 REM
16300 REM
16400 REM          LIST CURRENT PARAMETER VALUES
16500 GOSUB 17100
16600 RETURN
16700 REM
16800 REM
16900 REM *****
17000 REM
17100 REM          LIST CURRENT PARAMETER VALUES
17200 REM
17300 PRINT
17400 PRINT "CURRENT PARAMETER VALUES: 1. R1 = ";R1;" OHMS"
17500 PRINT "                2. L1 = ";CSNG(L1);" HENRY"
17600 PRINT "                3. C1 = ";CSNG(C1*1E+12);" PF"
17700 PRINT "                4. CO = ";CSNG(CO*1E+12);" PF"
17800 PRINT "                5. CXTAL1 = ";CSNG(CX*1E+12);" PF"
17900 PRINT "                6. CXTAL2 = ";CSNG(CY*1E+12);" PF"
18000 PRINT "                7. AMPLIFIER GAIN MAGNITUDE = ";AV#
18100 PRINT "                8. FEEDBACK RESISTANCE = ";CSNG(RX*.001);" K-OHMS"
18200 PRINT "                9. OUTPUT RESISTANCE = ";CSNG(RO*.001);" K-OHMS"
18300 PRINT
18400 PRINT "TO CHANGE A PARAMETER VALUE, TYPE (PARAM NO.),(NEW VALUE)."
18500 PRINT "OTHERWISE, TYPE 0.0."
18600 INPUT NX,X
18700 IF NX=0 THEN RETURN
18800 IF NX=1 THEN R1 = X
18900 IF NX=2 THEN L1 = X
19000 IF NX=3 THEN C1 = X*1E-12
19100 IF NX=4 THEN CO = X*1E-12
19200 IF NX=5 THEN CX = X*1E-12
19300 IF NX=6 THEN CY = X*1E-12
19400 IF NX=7 THEN AV# = X
19500 IF NX=8 THEN RX = X*1000#
19600 IF NX=9 THEN RO = X*1000#
19700 GOTO 17400
19800 REM
19900 REM
20000 REM *****
20100 REM
20200 REM          CIRCUIT ANALYSIS
20300 REM
20400 REM          This routine calculates the loop gain at complex frequency SQ+jFQ.
20500 REM
20600 REM          1. Crystal impedance: RE + jXE
20700 REM
20800 X1 = FNXL(FG,L1) + FNXC(SG,FG,C1)
20900 RE = FNRP((R1+FNRL(SG,L1)+FNRC(SG,FG,C1)),X1,FNRC(SG,FG,CO),FNXC(SG,FG,CO))
21000 XE = FNXP((R1+FNRL(SG,L1)+FNRC(SG,FG,C1)),X1,FNRC(SG,FG,CO),FNXC(SG,FG,CO))
21100 REM
21200 REM          2. RF + jXF = (RE+jXE)/((amplifier feedback resistance)
21300 REM
21400 RF = FNRP(RX,0,RE,XE)
21500 XF = FNXP(RX,0,RE,XE)
21600 REM
21700 REM          3. Input impedance: Zi = RI + jXI = impedance of CXTAL1
21800 REM
21900 RI = FNRC(SG,FG,CX)
22000 XI = FNXC(SG,FG,CX)
22100 REM
22200 REM          4. Load impedance: ZL = (impedance of CXTAL2)/((RF+RI)+j(XF+XI))
22300 REM
22400 RL = FNRP((RF+RI),(XF+XI),FNRC(SG,FG,CY),FNXC(SG,FG,CY))
22500 XL = FNXP((RF+RI),(XF+XI),FNRC(SG,FG,CY),FNXC(SG,FG,CY))
22600 REM
22700 REM          5. Amplifier gain: A = -AV*ZL/(ZL+RO)
22800 REM          = A(real) + jA(imaginary)
22900 REM
23000 AR# = -AV*FNRP(RL,XL,(RO+RL),XL)
23100 AI# = -AV*FNXP(RL,XL,(RO+RL),XL)
23200 REM
23300 REM          6. Feedback ratio: (beta) = (RI+jXI)/((RF+RI)+j(XF+XI))
23400 REM          = B(real) + jB(imaginary)

```

```

23500 REM
23600 BR# = FNRR(RI, XI, (RI+RF), (XI+XF))
23700 BI# = FNXR(RI, XI, (RI+RF), (XI+XF))
23800 REM
23900 REM 7. Amplifier gain in magnitude/phase form: AR+jAI = A at AP degrees
24000 REM
24100 A = FNZM(AR#, AI#)
24200 AP = FNZP(AR#, AI#)
24300 REM
24400 REM 8. (beta) in magnitude/phase form: BR+jBI = B at BP degrees
24500 REM
24600 B = FNZM(BR#, BI#)
24700 BP = FNZP(BR#, BI#)
24800 REM
24900 REM 9. Loop gain: G = (BR+jBI)*(AR+jAI)
25000 REM      G = G(real) + jG(imaginary)
25100 REM
25200 GR = FNRM(AR#, AI#, BR#, BI#)
25300 GI = FNXM(AR#, AI#, BR#, BI#)
25400 REM
25500 REM 10. Loop gain in magnitude/phase form: GR+jGI = AL at AQ degrees
25600 REM
25700 AL = FNZM(GR, GI)
25800 AQ = FNZP(GR, GI)
25900 RETURN
26000 REM
26100 REM
26200 REM *****
26300 REM
26400 REM      PRINT CIRCUIT ANALYSIS RESULTS
26500 REM
26600 PRINT
26700 PRINT " FREQUENCY = "; SQ; " + J"; FQ; " HZ"
26800 PRINT " XTAL IMPEDANCE = "; FNZM(RE, XE); " OHMS AT "; FNZP(RE, XE); " DEGREES"
26900 PRINT "      (RE = "; CSNG(RE); " OHMS)"
27000 PRINT "      (XE = "; CSNG(XE); " OHMS)"
27100 PRINT " LOAD IMPEDANCE = "; FNZM(RL, XL); " OHMS AT "; FNZP(RL, XL); " DEGREES"
27200 PRINT " AMPLIFIER GAIN = "; A; " AT "; AP; " DEGREES"
27300 PRINT " FEEDBACK RATIO = "; B; " AT "; BP; " DEGREES"
27400 PRINT " LOOP GAIN = "; AL; " AT "; AQ; " DEGREES"
27500 RETURN
27600 REM
27700 REM
27800 REM *****
27900 REM
28000 REM      SEARCH FOR FREQUENCY (S+JF)
28100 REM      AT WHICH LOOP GAIN HAS ZERO PHASE ANGLE
28200 REM
28300 REM This routine searches for the frequency at which the imaginary part
28400 REM of the loop gain is zero. The algorithm is as follows:
28500 REM 1. Calculate the sign of the imaginary part of the loop gain (GI)
28600 REM 2. Increment the frequency.
28700 REM 3. Calculate the sign of GI at the incremented frequency.
28800 REM 4. If the sign of GI has not changed, go back to 2.
28900 REM 5. If the sign of GI has changed, and this frequency is within
29000 REM 1Hz of the previous sign-change, exit the routine.
29100 REM 6. Otherwise, divide the frequency increment by -10.
29200 REM 7. Go back to 2.
29300 REM The routine is entered with the starting frequency SQ+jFQ and
29400 REM starting increment DS+jDF already defined by the calling program.
29500 REM In actual use either DS or DF is zero, so the routine searches for
29600 REM a GI=0 point by incrementing either SQ or FQ while holding the other
29700 REM constant. It returns control to the calling program with the
29800 REM incremented part of the frequency being within 1Hz of the actual
29900 REM GI=0 point.
30000 REM
30100 REM 1. CALCULATE THE SIGN OF THE IMAGINARY PART OF THE LOOP GAIN (GI).
30200 REM
30300 GOSUB 20200
30400 GOSUB 26600
30500 IF GI=0 THEN RETURN
30600 SX% = INT(SGN(GI))
30700 IF SX%=-1 THEN DS = -DS
30800 REM (REVERSAL OF DS FOR GI<0 IS FOR THE POLE-SEARCH ROUTINE.)
30900 REM
31000 REM 2. INCREMENT THE FREQUENCY.
31100 REM
31200 SP = SQ

```



```

31300 FP = FQ
31400 SQ = SQ + DS
31500 FQ = FQ + DF
31600 REM
31700 REM 3. CALCULATE THE SIGN OF GI AT THE INCREMENTED FREQUENCY.
31800 REM
31900 GOSUB 20200
32000 GOSUB 26600
32100 IF INT(SGN(GI))=0 THEN RETURN
32200 REM
32300 REM 4. IF THE SIGN OF GI HAS NOT CHANGED, GO BACK TO 2.
32400 REM
32500 IF SXZ+INT(SGN(GI))=0 THEN PRINT ELSE 31400
32600 SXZ = -SXZ
32700 REM
32800 REM 5. IF THE SIGN OF GI HAS CHANGED, AND IF THIS FREQUENCY IS WITHIN
32900 REM 1HZ OF THE PREVIOUS SIGN-CHANGE, AND IF GI IS NEGATIVE, THEN
33000 REM EXIT THE ROUTINE. (THE ADDITIONAL REQUIREMENT FOR NEGATIVE GI
33100 REM IS FOR THE POLE-SEARCH ROUTINE.)
33200 REM
33300 IF ABS(SP-SQ)<1 AND ABS(FP-FQ)<1 AND SXZ=-1 THEN RETURN
33400 REM
33500 REM 6. DIVIDE THE FREQUENCY INCREMENT BY -10.
33600 REM
33700 DS = -DS/10#
33800 DF = -DF/10#
33900 REM
34000 REM 7. GO BACK TO 2.
34100 REM
34200 GOTO 31200
34300 REM
34400 REM
34500 REM *****
34600 REM SEARCH FOR POLE FREQUENCY
34700 REM
34800 REM
34900 REM This routine searches for the frequency at which the loop gain = 1
35000 REM at 0 degrees. That frequency is the pole frequency of the closed-
35100 REM loop gain function. The pole frequency is a complex number, SQ+jFQ
35200 REM (Hz). Oscillator start-up ensues if SQ>0. The algorithm is based on
35300 REM the calculated behavior of the phase angle of the loop gain in the
35400 REM region of interest on the complex plane. The locus of points of zero
35500 REM phase angle crosses the j-axis at the oscillation frequency and at
35600 REM some higher frequency. In between these two crossings of the j-axis,
35700 REM the locus lies in Quadrant I of the complex plane, forming an
35800 REM approximate parabola which opens to the left. The basic plan is to
35900 REM follow the locus from where it crosses the j-axis at the oscillation
36000 REM frequency, into Quadrant I, and find the point on that locus where
36100 REM the loop gain has a magnitude of 1. The algorithm is as follows:
36200 REM 1. Find the oscillation frequency, 0+jFQ.
36300 REM 2. At this frequency calculate the sign of (AL-1). (AL = magnitude
36400 REM of loop gain.)
36500 REM 3. Increment FQ.
36600 REM 4. For this value of FQ, find the value of SQ for which the loop
36700 REM gain has zero phase.
36800 REM 5. For this value of SQ+jFQ, calculate the sign of (AL-1).
36900 REM 6. If the sign of (AL-1) has not changed, go back to 3.
37000 REM 7. If the sign of (AL-1) has changed, and this value of FQ is
37100 REM within 1Hz of the previous sign-change, exit the routine.
37200 REM 8. Otherwise, divide the FQ-increment by -10.
37300 REM 9. Go back to 3.
37400 REM
37500 REM 1. FIND THE OSCILLATION FREQUENCY, 0+jFQ.
37600 REM
37700 GOSUB 9700
37800 GOSUB 30300
37900 REM
38000 REM 2. AT THIS FREQUENCY, CALCULATE THE SIGN OF (AL-1).
38100 REM
38200 SYZ = INT(SGN(AL-1))
38300 IF SYZ=-1 THEN STOP
38400 REM ESTABLISH INITIAL INCREMENTATION VALUE FOR FQ.
38500 F1 = FQ
38600 DF = (FA-F1)/10#
38700 GOSUB 30300
38800 DE = (FQ-F1)/10#
38900 DF = 0
39000 FQ = F1

```



```

39100 REM
39200 REM 3. INCREMENT FQ.
39300 REM
39400 FQ = FQ + DE
39500 REM
39600 REM 4. FOR THIS VALUE OF FQ, FIND THE VALUE OF SQ FOR WHICH THE LOOP
39700 REM GAIN HAS ZERO PHASE. (THE ROUTINE WHICH DOES THAT NEEDS DF = 0,
39800 REM SO THAT IT CAN HOLD FQ CONSTANT, AND NEEDS AN INITIAL VALUE FOR
39900 REM DS, WHICH IS ARBITRARILY SET TO DS = 1000.)
40000 REM
40100 DS = 1000#
40200 SQ = 0
40300 GOSUB 30300
40400 IF AL=1! THEN RETURN
40500 REM
40600 REM 5. FOR THIS VALUE OF SQ+JFQ, CALCULATE THE SIGN OF (AL-1).
40700 REM 6. IF THE SIGN OF (AL-1) HAS NOT CHANGED, GO BACK TO 3.
40800 REM
40900 IF SYX+INT(SGN(AL-1))=0 THEN PRINT ELSE 39400
41000 REM
41100 REM 7. IF THE SIGN OF (AL-1) HAS CHANGED, AND THIS VALUE OF FQ IS WITHIN
41200 REM 1HZ OF THE PREVIOUS SIGN-CHANGE, EXIT THE ROUTINE.
41300 REM
41400 IF ABS(F1-FQ)<1 THEN RETURN
41500 REM
41600 REM 8. DIVIDE THE FQ-INCREMENT BY -10.
41700 REM
41800 DE = -DE/10#
41900 F1 = FQ
42000 SYX = -SYX
42100 REM
42200 REM 9. GO BACK TO 3.
42300 REM
42400 GOTO 39400
42500 REM
42600 REM
42700 REM *****
42800 REM
42900 REM STEADY-STATE ANALYSIS
43000 REM
43100 REM The circuit model used in this analysis is similar to the one used
43200 REM in the small-signal analysis, but differs from it in two respects.
43300 REM First, it includes clamping and clipping effects described in the
43400 REM text. Second, the voltage source in the Thevenin equivalent of the
43500 REM amplifier is controlled by the input voltage in accordance with an
43600 REM input-output curve defined elsewhere in the program.
43700 REM The analysis applies a sinusoidal input signal of arbitrary
43800 REM amplitude, at the oscillation frequency, to the XTAL1 pin, then
43900 REM calculates the resulting waveform from the voltage source. Using
44000 REM standard Fourier techniques, the fundamental frequency component of
44100 REM this waveform is extracted. This frequency component is then
44200 REM multiplied by the factor  $1/ZL/(ZL+RO)$ , and the result is taken to be
44300 REM the signal appearing at the XTAL2 pin. This signal is then
44400 REM multiplied by the feedback ratio (beta), and the result is taken to
44500 REM be the signal appearing at the XTAL1 pin. The algorithm is now
44600 REM repeated using this computed XTAL1 signal as the assumed input
44700 REM sinusoid. Every time the algorithm is repeated, new values appear at
44800 REM XTAL1 and XTAL2, but the values change less and less with each
44900 REM repetition. Eventually they stop changing. This is the steady-state.
45000 REM The algorithm is as follows:
45100 REM 1. Compute approximate oscillation frequency.
45200 REM 2. Call a circuit analysis at this frequency.
45300 REM 3. Find the quiescent levels at XTAL1 and XTAL2 (to establish the
45400 REM beginning DC level at XTAL1).
45500 REM 4. Assume an initial amplitude for the XTAL1 signal.
45600 REM 5. Correct the DC level at XTAL1 for clamping effects, if necessary.
45700 REM 6. Using the appropriate input-output curve, extract a DC level and
45800 REM the fundamental frequency component (multiplying the latter by
45900 REM  $1/ZL/(ZL+RO)$ ).
46000 REM 7. Clip off the negative portion of this output signal, if the
46100 REM negative peak falls below zero.
46200 REM 8. If this signal, multiplied by (beta), differs from the input
46300 REM amplitude by less than 1mV, or if the algorithm has been repeated
46400 REM 10 times, exit the routine.
46500 REM 9. Otherwise, multiply the XTAL2 amplitude by (beta) and feed it
46600 REM back to XTAL1, and go back to 5.
46700 REM
46800 REM 1. COMPUTE APPROXIMATE OSCILLATION FREQUENCY.

```

```

46900 GOSUB 9700
47000 REM
47100 REM 2. CALL A CIRCUIT ANALYSIS AT THIS FREQUENCY.
47200 GOSUB 20800
47300 PRINT : PRINT : PRINT "ASSUMED OSCILLATION FREQUENCY:"
47400 GOSUB 26600
47500 PRINT : PRINT
47600 REM
47700 REM 3. FIND QUIESCENT POINT
47800 REM (At quiescence the voltages at XTAL1 and XTAL2 are equal. This
47900 REM voltage level is found by trial-and-error, based on the input-
48000 REM output curve, so that a person can change the input-output curve
48100 REM as desired without having to re-calculate the quiescent point.)
48200 VI = 0
48300 VB = 1
48400 K1 = 1
48500 VI = VI + VB
48600 GOSUB 13600
48700 IF ABS(V0-VI)<.001 THEN 49200
48800 IF K1+SGN(V0-VI)=0 THEN 48900 ELSE 48500
48900 K1 = SGN(V0-VI)
49000 VB = -VB/10
49100 GOTO 48500
49200 VB = VI
49300 PRINT "QUIESCENT POINT = ";VB
49400 REM
49500 REM 4. ASSUME AN INITIAL AMPLITUDE FOR THE XTAL1 SIGNAL.
49600 EI = .01
49700 NRX = 0
49800 REM
49900 REM 5. CORRECT FOR CLAMPING EFFECTS, IF NECESSARY.
50000 REM (K1 and K2 are curve-fitting parameters for the RDM parts.)
50100 K1 = (2.5-VB)/(3-VB)
50200 K2 = (VB-1.25)/(3-VB)
50300 IF ICX=2 OR ICX=4 THEN IF EI<(VB+.5) THEN EO = VB ELSE EO = EI -.5
50400 IF ICX=1 OR ICX=3 THEN IF EI<(VB+.5) THEN EO = VB ELSE EO = K1*EI+K2
50500 NRX = NRX + 1
50600 REM
50700 REM 6. DERIVE XTAL2 AMPLITUDE.
50800 V0 = 0
50900 VC = 0
51000 VS = 0
51100 FOR NX = -25 TO +24
51200 VI = EO - EI*COS(PI*NX/25)
51300 GOSUB 13600
51400 V0 = V0 + VI
51500 VC = VC + V0*COS(PI*NX/25)
51600 VS = VS + V0*SIN(PI*NX/25)
51700 NEXT NX
51800 V0 = V0/50
51900 VI = SQR(V0^2+VS^2)/25*FNZM(RL,XL)/FNZM((RL+RD),XL)
52000 REM
52100 REM 7. CLIP XTAL2 SIGNAL.
52200 IF V0-VI<0 THEN VL = 0 ELSE VL = V0-VI
52300 PRINT : PRINT "XTAL1 SWING = ";EO-EI;" TO ";EO+EI
52400 PRINT "XTAL2 SWING = ";VL;" TO ";V0+VI
52500 REM
52600 REM 8. TEST FOR TERMINATION.
52700 IF ABS(EI-VI*B)<.001 OR NRX=10 THEN RETURN
52800 REM
52900 REM 9. FEED BACK TO XTAL1 AND REPEAT
53000 EI = VI*B
53100 GOTO 50300

```